

A DRAFTING TOOL FOR OBJECT - ORIENTED ANALYSIS : DESIGN AND IMPLEMENTATION

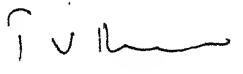
*A Thesis Submitted
In Partial Fulfilment of the Requirements
for the Degree of*
MASTER OF TECHNOLOGY

By
SAYA SREENIVASULU

to the
**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY KANPUR
MAY, 1992**

CERTIFICATE

It is certified that the work contained in the thesis entitled "A DRAFTING TOOL FOR OBJECT-ORIENTED ANALYSIS : DESIGN AND IMPLEMENTATION", by "SAYA SREENIVASULU", has been carried out under my supervision and that this work has not been submitted elsewhere for a degree.


(T.V. Prabhakar)
(Asst. Professor)
Dept. of CSE.
IIT. Kanpur.

May, 1992.

28 AUG 1992

CENTRAL LIBRARY
111 KANPUR

Acc. No. AJJ.4058

Th

005.15

Sr 18d

CSE-1992-M-SRE-DRA

ABSTRACT

This thesis presents a design and implementation of a drafting tool, "Object-Oriented Analysis Tool(OOATool)", for Object-Oriented Analysis. A user-friendly interface is designed to aid the user to analyze a system and build it's schema using OOA notation. The OOATool is implemented in C++, a block structured object-oriented language based on ANSI C language.

The tool has a comprehensive set of operations to edit the OOA schema. It generates analysis document of the schema as required by the designer. Presently, it generates C++ class templates for the OOA schema. The main features of the OOATool are: OOA schema editing, scrolling in all four directions, browsing, consistency checking and precise error reporting.

ACKNOWLEDGEMENTS

I express my heartfelt thanks to Dr. T.V. Prabhakar for his expert guidance and encouragement during his supervision of this thesis work. His suggestions and critical remarks were very much helpful during the development and the preparation of this thesis.

I am also thankful to Ved Prakash Sinha for his patience for sitting with me the whole night and helping me in drawing figures. Especially he deserves thanks from all of our other H-Topites for making us to enjoy "food-items" he offers now-and-then. Just any house-wife can't beat him. Really Y. Krishna Bhargava proved himself as a good friend of mine with his genuine helping nature when I was in 'tough-time'.

Next my thanks goes to other H-Topites, TVLN Siva Kumar, M.V. Rao and D.M. Mahendra who are always there for a friendly chit-chat. I must express my thanks to all my friends and specially Dr. D. Sunder Rajan for his discussions with me which made me to look at 'Life' from a new perspective.

Of course, I must particularly thank Software Engineer Mr. N.V. Bramhaji Rao who indirectly helped me to complete this thesis in time. Also N.K. Saxena and K. Vasudev Varma deserve thanks for proof-reading the draft of this thesis and PV Ravi Shankar deserve special mentioning for typing in the figures with proper indentation.

Lastly, I should thank Prof. R.N. Biswas, ACES and Motorola Inc. for supporting me financially during the crucial stage of this thesis development work.

*The Work in This Thesis
is Dedicated to My Parents*

**S. Nagaraju
and
S. Parvathi.**

CONTENTS

	Page No.
1. Introduction	1
1.1. Introduction	1
1.2. Features of Object-Oriented Analysis Tool	2
1.3. Related Work	3
1.4. System Architecture	4
1.5. Thesis Organization	5
2. Object-Object Analysis	6
2.1. Introduction	6
2.2. Analysis Methods	7
2.3. Object-Oriented Analysis Approach	10
2.3.1. Class&Object Layer	11
2.3.2. Structure Layer	12
2.3.3. Subject Layer	14
2.3.4. Attribute Layer	16
2.3.5. Service Layer	18
2.4. More about Each Layer	20
2.5. A Case Study	21
3. OOATool - User Interface	32
3.1. Introduction	32
3.2. User Interface Description	33
4. Design and Implementation	39
4.1. Object-Oriented Design - An Introduction	39

4.2. DBATool - Design and Implementation	42
5. Conclusions and Future Work	48
References	50
Appendix	
Appendix-A. User's Manual	53
Appendix-B. Listings of C++ class implementations	62
Appendix-C. A Sample Schema Store File	88
Appendix-D. A Sample Schema Information File	90
Appendix-E. A Sample Schema Templates File	92
Appendix-F. A Sample Schema Document File	94

LIST OF FIGURES

Figure No.	Page No.
1.1. Architecture of OOATool	4
2.1. Merging of disciplines	10
2.2. The multi-layer model	11
2.3a. The OOA "Class" symbol	12
2.3b. The OOA "Class&Object" symbol	12
2.4. The Gen-Spec structure notation	13
2.5. The Whole-Part structure notation	14
2.6a. Subject notation, collapsed	15
2.6b. Subject notation, partially expanded	15
2.7. Placing attributes in a class&object	16
2.8. Instance connection notation	17
2.9. Placing services in a class&object	19
2.10a. Message connection to single object	19
2.10b. Message connection to many objects	20
2.11. The functional decomposition for students registration system	22
2.12. The DFD for students registration system	23
2.13. The data dictionary for students registration system	24
2.14. The entity-relationship diagram for students registration system	25
2.15. The students registration system - Class&Object, Structure, Attribute and Service layers	26
2.16a. The subjects in students registration system	27

2.16b. The partially expanded subject in students registration system	27
2.17. The Classes and Class&Objects in students registration system	28
2.18. The Gen-Spec(hierarchy) structure in students registration system	29
2.19. The Attribute layer for students registration system	30
2.20. The Service layer in the students registration system	31
3.1. The screen layout of OOATool	33
3.2. The root menu and a sample sub-menu	34
3.3. A sample dialog window of OOATool	36
3.4. A sample info window of OOATool	37
4.1. Modeling Manager organization	43
4.2. Display Manager organization	44
4.3. Schema Manager organization	45
4.4. Drawing Sheet organization	46
A.1. The OOATool "Class" and "Class&Object" symbols	58
A.2a. The OOATool "Gen-Spec Hierarchy" structure notation	58
A.2b. The OOATool "Gen-Spec Lattice" structure notation	59
A.3. The OOATool "Instance" connection notation	59
A.4. The OOATool "Message" connection notation	59
A.5. Display of a class in OOATool	60

CHAPTER - 1

INTRODUCTION

1.1. Introduction

The methods and tools used during software development can have a significant impact on the quality of final product. Over the past 15 years or so, the "Structured Techniques" have gained a great deal of acceptance as being the method that can assist the analyst and designer in the system analysis and design phases respectively. This method is based on a functional view of the system, with the system being partitioned according to its functional aspects.

Recently a new approach to system analysis, *Object-Oriented Analysis(OOA)*, has been gaining popularity. OOA maps the problem domain directly into a model. Each object in the problem domain is mapped to an object in the model with similar behavior. The complexity of the problem domain is expressed as a set of classification structures and/or a set of assembly structures. Any similarity between a set of objects is captured in a classification structure. OOA also models the associations and interactions between objects.

In the analysis process, the structure and functioning of system under consideration is understood and is formally

documented. The output of analysis process is a precise formal specification of the system which is passed as input to design phase. This thesis is an attempt to build an user-friendly analysis system supporting Object-Oriented Analysis. The design of the system conforms to the design principles of object-oriented development. More over, the implementation of the system is done in an object-oriented language to fully exploit the benefits of object-oriented design.

1.2. Features of Object-Oriented Analysis Tool(OOATool)

The following are the features of our OOATool:

- i) Schema Presentation: A concise and consistent notation is used to present the OOA schema. The concept of virtual drawing sheet enables the user to enter arbitrarily large schema. A set of editing facilities allows the user to perform wide range of editing operations on the OOA schema.
- ii) Support for Incremental Analysis: The user can do his analysis incrementally, i.e., he can switch freely between different stages of analysis(i.e., on layer by layer basis), without doing full depth analysis of one stage(layer) at a time.
- iii) Browsing: The user can browse the classes in the schema, attributes and services of a class. He can also see constraints on an instance connection or

a whole-part structure of a class(instance connection and whole-part structure are discussed in chapter-2).

- iv) Consistency: The system allows the user to enter only correct and consistent schema. Any operation leading to inconsistency does not pass through the system. The conflicts in naming of objects, its attributes and services, which cause schema to be inconsistent, are reported and the operation which caused the conflict is not done. Similarly the operation which cause the schema to be incorrect are denied by the system.

1.3. Related Work

In past few years, many tools for systems analysis have been introduced into the commercial market. These tools are supplied by vendors like Ed Yourdon(for Yourdon method), Michael Jackson(for Jackson method) and Oracle corporation(for OracleCASE) and others. The commercially available tools like Excelerator, TurboAnalyst and TurboCASE support Structured Analysis.

Recently, the developers of Object Oriented Analysis(OOA), Peter Coad and Ed Yourdon are promoting OOA methodology[Coad and Yourdon, 1991a], [Shlaer and Mellor, 1988]. Presently, some commercially available CASE tools supporting OOA are Object-Oriented Environment, OOATool and

ObjectPlus.

Other tools might be in progress.

1.4. System Architecture

The architecture of OOATool is shown in Figure 1.1. The user interacts with the Interface Manager. The Modeling Manager communicates with Interface Manager to get input from the user. When the user requests for an operation on the schema, Modeling Manager sends a message to Schema Manager to effect the operation in the schema; and a message to Display Manager to display the operation that is done on the screen.

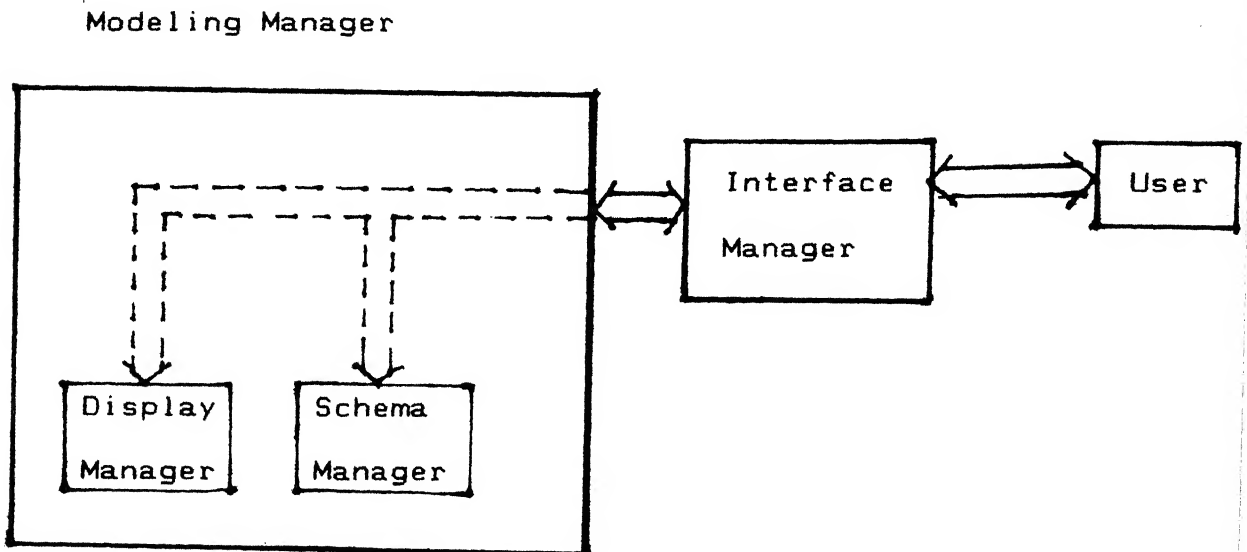


Figure 1.1. Architecture of OOATool.

1.5. Thesis Organization

This thesis is organized as follows. Chapter-2 gives a brief introduction of analysis methods, object-oriented analysis terms and concepts. Chapter-3 describes the user-interface of the system. Chapter-4 explains briefly object-oriented design methodology and the design of our OOATool.

Chapter-5 presents the conclusions and further work that can be done to OOATool to make it a complete and usable tool. The Appendix contains C++ class implementations of different modules of the tool, user manual, and sample files generated by OOATool.

CHAPTER - 2

OBJECT-ORIENTED ANALYSIS

2.1 Introduction

Object-Oriented Analysis(OOA) is based upon concepts like objects and attributes, wholes and parts, classes and members. The approach of OOA builds upon three constantly employed methods of organization. They are:

- 1) the differentiation of experience into particular objects and their attributes -- e.g., how do we distinguish a tree by its size and/or its spatial relation to other objects.
- 2) the distinction between whole objects and their component parts -- e.g., how do we contrast a tree with its component branches and leaves, and
- 3) the formation of and the distinction between different classes of objects -- e.g., how do we distinguish class trees from class of all stones.

Benefits of OOA are as follows:

1. **Explicitly Represent Commonality:** OOA uses inheritance to identify and capitalize on commonality of Attributes and Services.

2. **Build Specifications Resilient to Change:** OOA embeds volatility within problem domain constructs, providing stability over changing requirements.
3. **Reuse Analysis Results:** OOA organizes results based upon problem domain constructs for present and future reuse.
4. **Provide a Consistent Underlying Representation for Analysis(what is to be built) and design(how it is to be built this time):** OOA establishes a continuum of representation for systematically expanding analysis results into a specific design.

2.2 Analysis Methods

DeMacro offers the following definition for systems analysis [DeMacro, 1978]: "Analysis is the study of a problem(system), prior to taking some action regarding the problem(system)". In other words, analysis is the study of a problem domain, leading to a specification of externally observable behavior; a complete, consistent, and feasible statement of what is needed.

This section surveys four major approaches to analysis. These approaches are thinking tools, used to help in the formulation of requirements. We illustrate each of these methodologies through an example problem later in this chapter(in Section 2.5).

1) **Functional Decomposition** : This method has following three components:

- * *Functions*
- * *Sub-functions*
- * *Functional Interfaces.*

This method requires one to map problem-domain(e.g., student registration system) to functions and sub-functions[Yourdon and Constantine, 1978]. The analysis documentation consists of required functions and sub-functions that the system shall provide.

2) **Data-flow Approach** :This method is well-known as Structured Analysis. The data-flow approach has following components:

- * *Data(and Control) flows*
- * *Data transformations*
- * *Data stores*
- * *Process Specifications*
- * *Data Dictionary.*

In this approach, the flow of data and the processes from/to which the data flows are identified and drawn. There are two major strategies are popular in structured analysis. The "old" method[DeMacro, 1978], [Gane and Sarson, 1977] maps the system to data-flow diagrams. The "modern" structured analysis approach[McMenamin and Palmer, 1982], [Yourdon, 1989] first identifies the events that occur in the outside world. these events are then transformed to a process or a set of processes. Each process is names by deciding what the system must do in response to the event.

3) **Information Modeling** :The information modeling combines the concepts from Entity-Relationship model and semantic data model[Chen, 1976], [Flavin, 1981],[Hammer and McLeod, 1981] and [Teorey et al., 1986]. This approach has following components:

- * *Objects*
- * *Attributes*
- * *Relationships*
- * *Supertype/Subtypes*
- * *Associative Objects*

This method has the following missing concepts:

- i) **Services**: describes the behavior of an object
- ii) **Messages**: a narrow, well-defined interface, depicting processing inter-dependency.
- iii) **Inheritance**: explicit representation of Attribute and Service commonality.
- iv) **Structure**: Whole-Part structures are not captured.

4) **Object-Oriented Approach** :The Object-Oriented approach has the following components:

- * *Class and Objects*
- * *Inheritance*
- * *Communication with messages*

The OOA builds upon the best concepts from Information Modeling, Object-Oriented Programming Languages, and Knowledge-Base Systems.

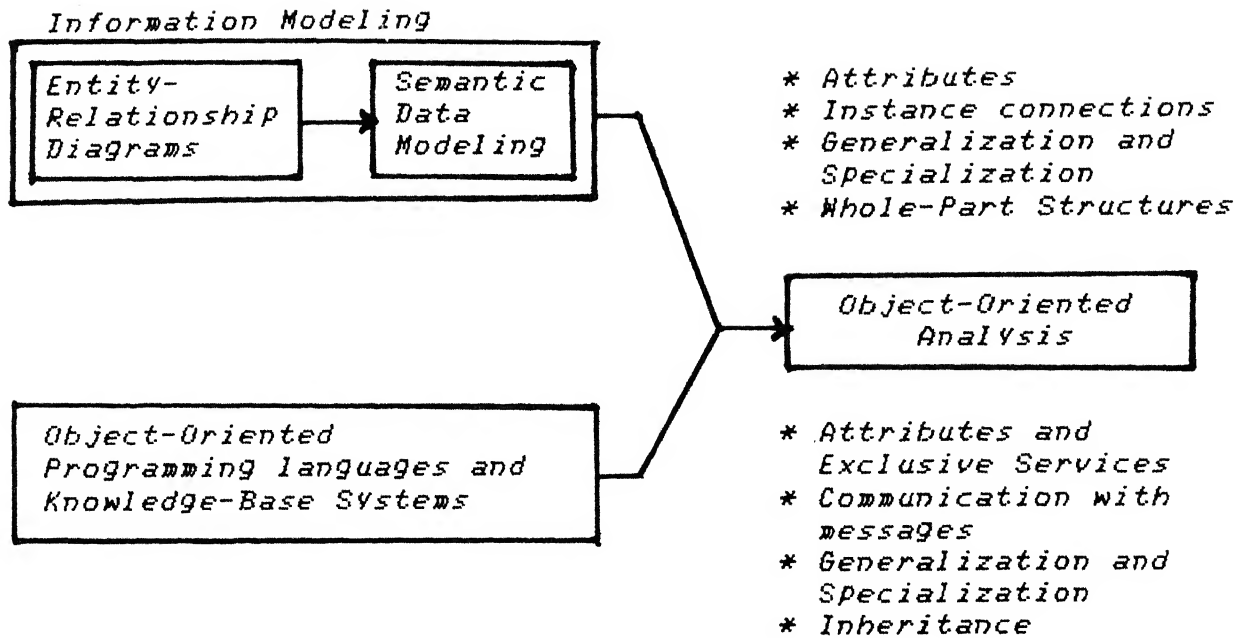


Figure 2.1: Merging of disciplines

A comparative study of object-oriented and structured developments can be found in [Patrick, 1990].

2.3. Object-Oriented Analysis Approach

In an overall approach, OOA consists of five major activities. They are:

- i) Finding Class&Objects
- ii) Identifying Structures
- iii) Identifying Subjects
- iv) Defining Attributes
- v) Defining Services

The OOA model is presented and reviewed in five layers:

```

----- Subject Layer
----- Class&Object Layer
----- Structure Layer
----- Attribute Layer
----- Service Layer

```

Figure 2.2: The multi-layer model

2.3.1. Class&Object Layer :

In this layer the Classes and Class&Objects that exist in the problem-domain are identified.

DEFINITIONS:

Class:

Class is a 'description' of one or more Objects with a uniform set of Attributes and Services.

Object:

Object is an 'abstraction' of something in problem domain, with independent existence.

Class&Object:

Class&Object refers to a Class which creates Objects existing in the problem-domain.

e.g.: In an Sensor System, one can identify two Class&Objects Sensor and Critical Sensor and no Classes.

Notation:

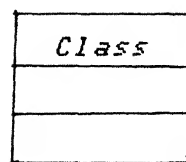


Figure 2.3a: The OOA "Class" symbol

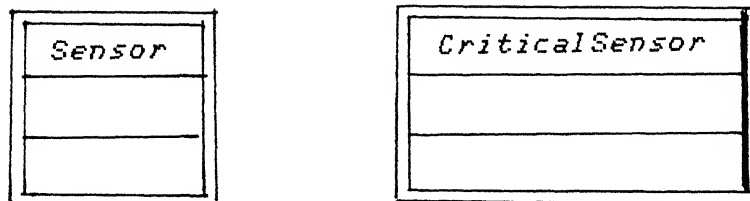


Figure 2.3b: The OOA "Class&Object" symbol

2.3.2. Structure Layer :

In this layer the Generalization-Specialization(Gen-Spec) and Whole-Part structures that exist in the system are identified.

DEFINITIONS:

Structure:

Structure is an expression of problem-domain complexity.

Generalization-Specialization Structure:

Generalization-Specialization(Gen-Spec) structure is a

hierarchical organization of a set of classes in the problem-domain. A Generalization-Specialization structure forms either a hierarchy or a lattice.

e.g.: In an organization, the following two Gen-Spec structures can be identified. In the following figure, Person, OwnerPerson and EmployerPerson form a Gen-Spec Hierarchy Structure and OwnerPerson, EmployerPerson and OwnerEmployerPerson form a Gen-Spec Lattice Structure.

Notation:

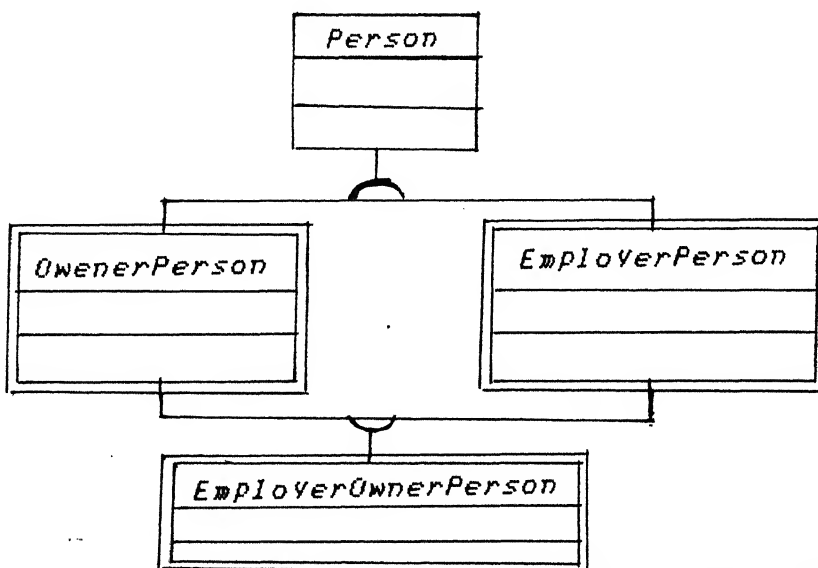


Figure 2.4: The Gen-Spec structure notation

Whole-Part Structure:

Whole-Part structure is an organization of classes showing container class and the contained classes.

e.g.: While analyzing a vehicle class, the following Whole-Part structure can be identified. In this structure, Vehicle is container class and Engine, Frame and Wheel are contained classes.

Notation:

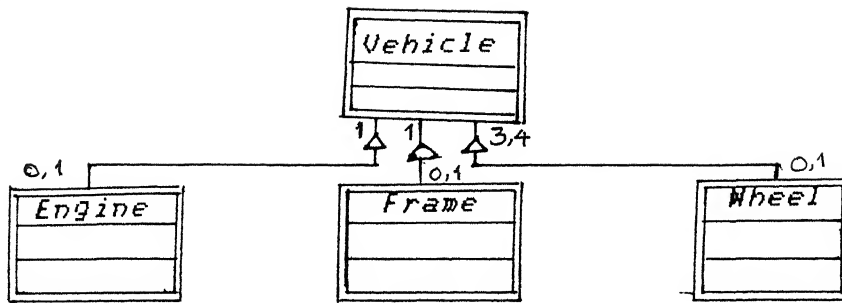


Figure 2.5: The Whole-Part structure notation

In the above example an engine is a part of:

- * possibly no vehicle
- * at most one vehicle

and a vehicle is an assembly of:

- * one engine
- * one frame
- * three or four wheels

2.3.3. Subject Layer :

This layer gives an overview of a large OOA model.

DEFINITIONS:

Subject:

A Subject is a mechanism for guiding a reader through a large and complex model.

Importance of Subjects:

In order to ease the comprehension of the problem-domain, two or more classes are to be grouped to form a higher level abstracted entity called Subject. Subjects are identified to control the amount of complexity confronted by the reader. Thus subject layer presents overall model from an higher perspective.

Notation:

1. Subject1

2. Subject2

Figure 2.6a: Subject notation, collapsed

1. Subject1	2. Subject2
1. Class&Object1	1. Class&Object3
2. Class&Object2	2. Class4
: : : :	: : : :
: : : .	: : : .

Figure 2.6b: Subject Notation, partially expanded(a CASE tool option)

2.3.4. Attribute Layer :

DEFINITIONS:

Attribute:

Attribute is some data(state information) for which each Object in a Class has its own value.

Importance of Attributes:

Attributes add detail to the Class&Object and Structure abstractions. Choosing Attributes involves analysis and one's personal choice. Attributes are placed in the center section of the Class&Object and Class symbols.

e.g.: For a Sensor Class&Object, some of its attributes can be identified as Model, Location, Threshold, State, Value.

Notation:

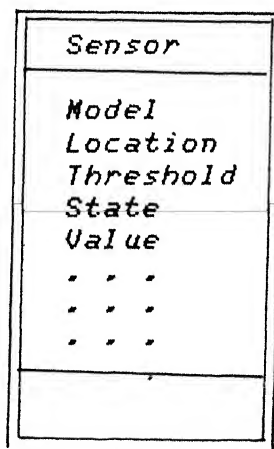


Figure 2.7: Placing attributes in a Class&Object

Attributes describe values(state) kept within an Object, to be exclusively manipulated by the Services of that Object(we will describe Services of Object shortly in

Service Layer). If another part of the system needs to access or otherwise manipulate the values in an object, it must do so by specifying a Message Connection (this also we will describe in Service Layer) corresponding to a Service defined for that Object.

Instance Connection:

Instance connection model an association between two objects.

If an Object needs association with another Object, this is represented by an Instance Connection between them. It is shown with a line drawn between the Objects. The Instance Connection is between two Objects (rather than between Classes). Each Object has amount(m) or range(m,n) markings on each of its Instance Connections, reflecting its constraints with other Objects.

e.g.: In an Institute system, the association between a student and a faculty member can be identified as:

Notation:

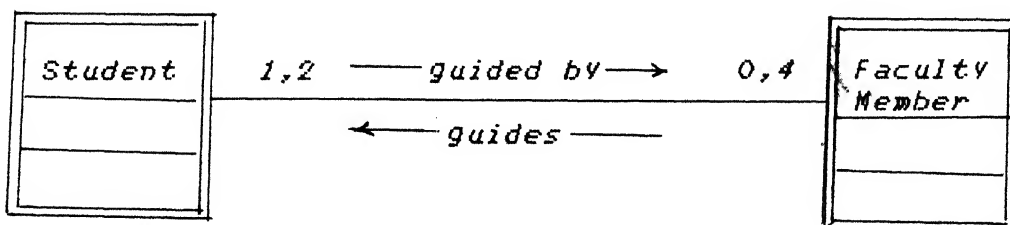


Figure 2.8: Instance connection notation

In the above figure, a student can be guided by one or two faculty members and each faculty member can guide zero or

four students.

Instance Connection can be labeled for better understanding, if the labels would not cause cluttering. Usually the label is avoided if the meaning of the association is clear from the context.

2.3.5. Service Layer :

DEFINITIONS:

Service:

Service is a specific behavior that an object exhibits when looked from outside the object.

Importance of Services:

Services further detail the abstraction of the reality being modeled, indicating what behavior will be provided by an Object within a Class. They also provide communication between Objects. Services are placed in the bottom section of the Class&Object and Class symbols.

e.g.: For a Sensor Class&Object, some of its attributes can be identified as Model, Location, Threshold, State, Value.

Notation:

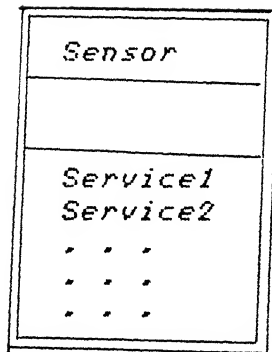


Figure 2.9: Placing services in a Class&Object

Message Connection:

Message Connection models the processing dependency of an Object, indicating a need for Service(s) in order to fulfill its responsibilities.

A Message Connection is a mapping of one Object to another Object, in which the "sender" sends a message to the "receiver" to get some processing done. The needed processing is named in the sender's services specification, and is defined in the receiver's service specification.

Notation:

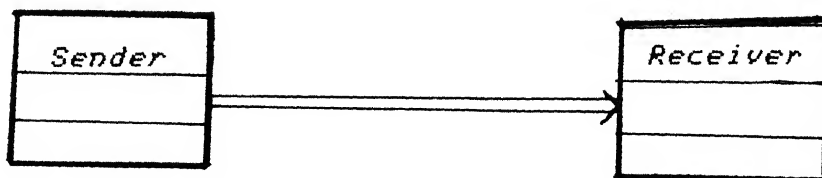


Figure 2.10a: Message connection to single object

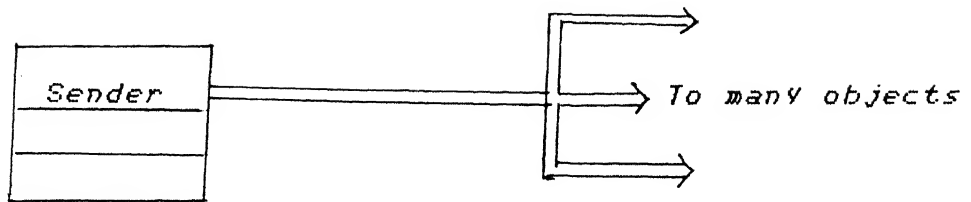


Figure 2.10b: Message connection to many objects

2.4. More about Each Layer:

In this chapter, we briefly described OOA concepts, terms, and notation. For detailed object-oriented analysis methodology of each layer, we request the reader to refer to [Coad & Yourdon, 91].

EXAMPLE : Students Registration System

This problem is analyzed using the four major approaches to analysis.

Problem-Domain Description:

In an Institute, students are enrolled every semester. All the students should register for a minimum number of credits. Every student will be guided by a faculty member(instructor). A faculty member can guide at most four students. An instructor can offer at most three courses. A student is allowed to register only if:

- 1). he cleared all dues to the institute, library and hostel,*
- 2). he secured the required minimum CPI in the last semester.*

A student is allowed to drop a course and add a new course with the consent of the concerned instructors.

Functional Decomposition : When the students registration system is viewed from functional view point, two main functions `Register_new_student()` and `Register_old_student()` are identified. The function `Register_new_student()` is decomposed into sub-functions `Validate_new_student()`, `check_for_dues()` and `Add_course()`. The `Add_course()` sub_function is further decomposed as shown in the following figure. The decomposition of the sub-function `Drop_course()` produces a new sub-function `Drop_course_for_student()`, in addition to having `Validate_course_no()` and `Validate_instructor()`. Two additional sub-functions `Validate_old_student()` and `check_for_required_CPI()` are identified when the function `Register_old_student()` is decomposed. The functions and sub-functions that are identified are shown as a tree in the figure 2.1.

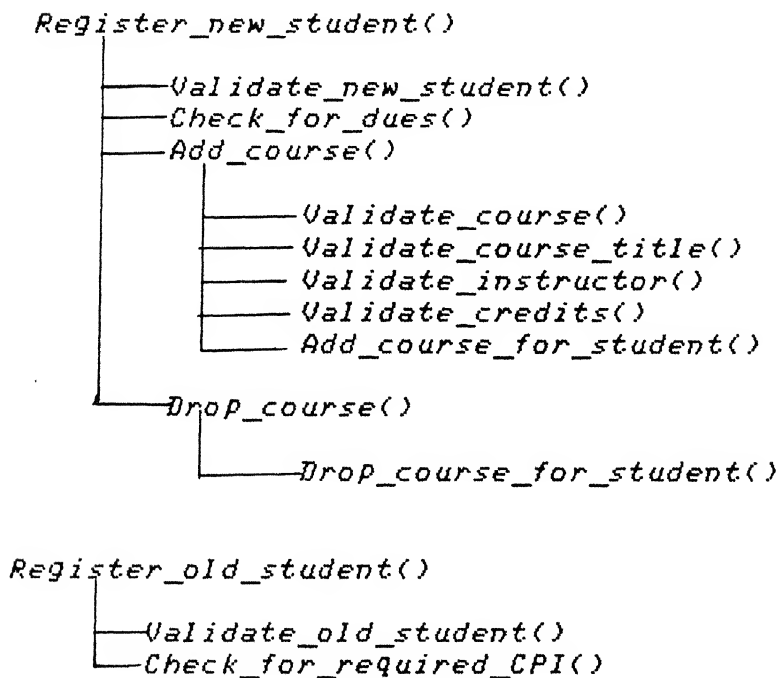


Figure 2.11: The Functional Decomposition for Student Registration System

Data flow Approach : The registration process of students registration system is shown in data-flow diagram (DFD) in figure 2.2. The DFD has the processes similar to the functions and sub-functions that are identified in functional decomposition. The major files in the system are: Students file, Dues file, Registered students file, Courses file and Student courses file.

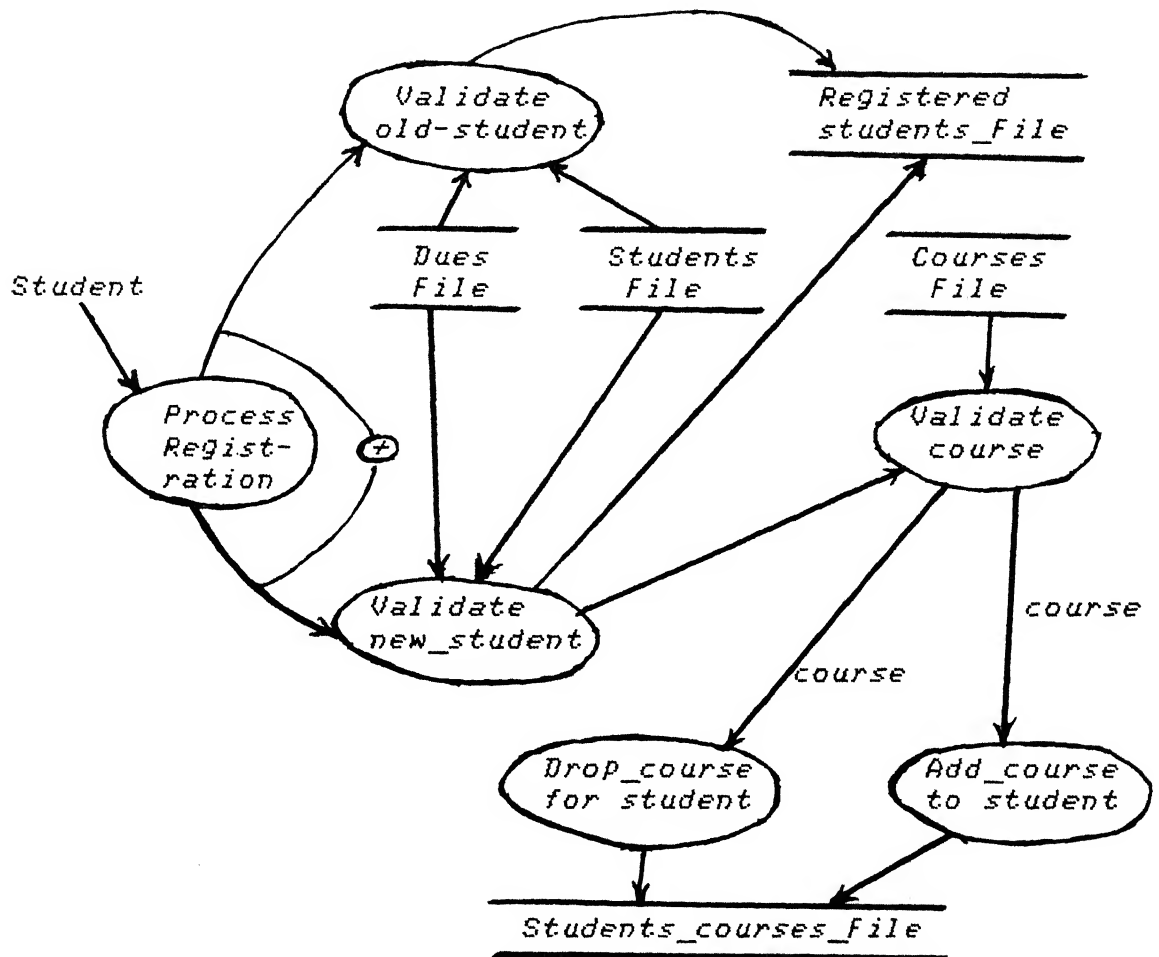


Figure 2.12: The DFD for Students Registration System

The data dictionary showing the structure and exact contents of each of the data-flow for students registration system is shown below.

```

Students_file = [ Roll_no + Student_name + Programme + Department
                  Address[ Hall_no + Room_no ] ]

Registered_students_file = [ Roll_no + Academic_Year + semester +
                              Registration_date ]

Courses_file = [ Course_no + Course_title + Units + Instructor_name
                 Instructors_initials ]

Course = [ Course_no + Units ]

Students_courses_file = [ Roll_no + [ Course_no + Registration_type
                                     + Thesis_supervisor ]

Registration_type = New / Repeat / Substitute

Dues_file = [ Roll_no + Due_to_whom + [ Due_amount / [Books]* ]* ]

```

Figure 2.13: The Data Dictionary for Student Registration System

Information Modeling : The analysis of students registration system through information modeling approach uncovers two new entities Person and Instructor. The model captures additional semantics of the real-world through IS_A hierarchy. The semantics of "an instructor guiding a student" and "an instructor offering a course" are also represented through relationships "guides" and "offers" respectively. the complete Entity-Relationship diagram is shown in the figure 2.4.

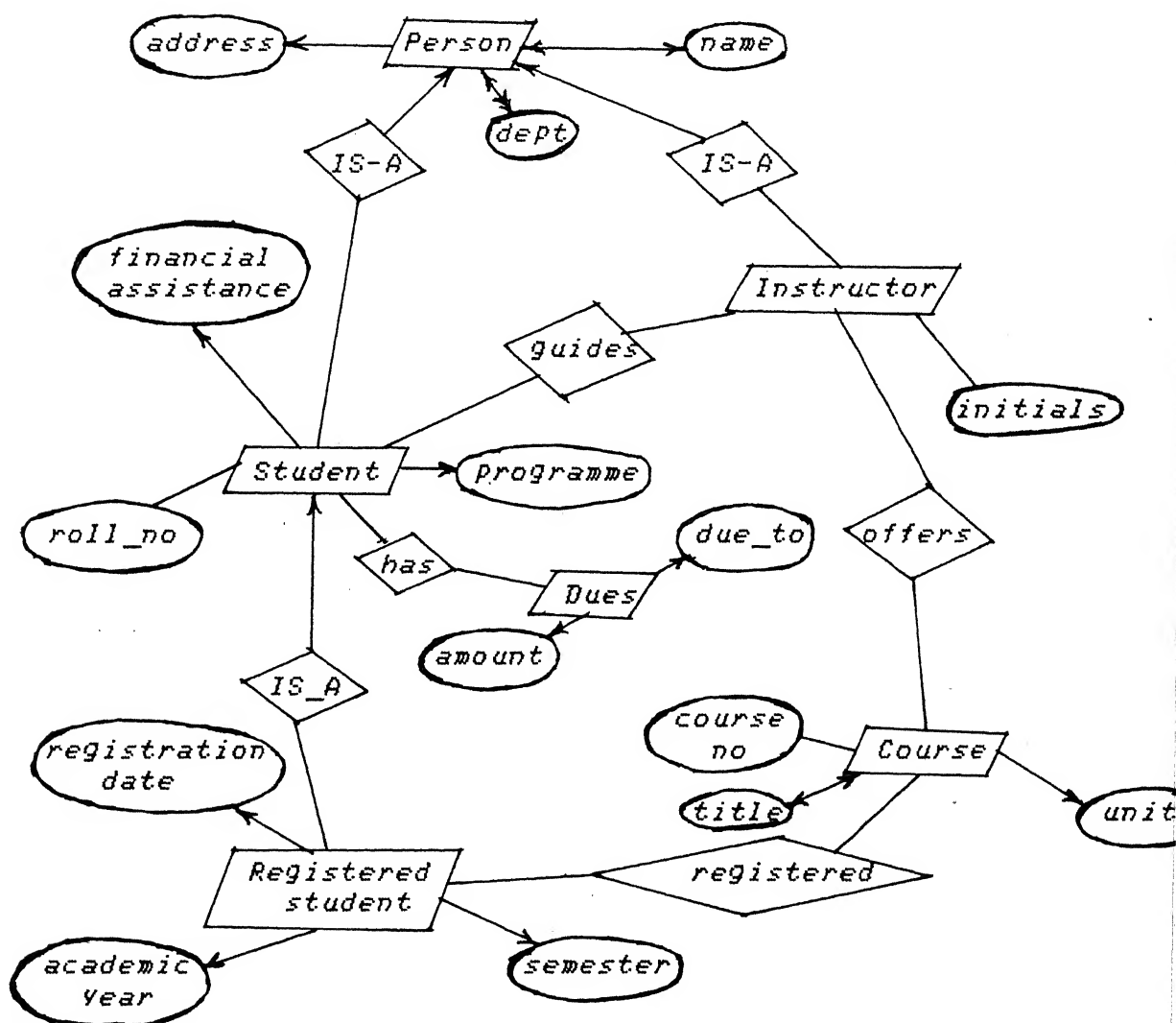


Figure 2.14: The Entity-relationship diagram for student registration system

Object-Oriented Approach : The object-oriented analysis approach identifies the complete behavior of all objects in the system through services explicitly. For example, the behavior of Registered_student is captured in the services Take courses, Pay dues and drop course. The constraints on the association between (Student, Instructor) and (Instructor, Course) are explicitly specified. The communication between objects is also shown as double line arrows.

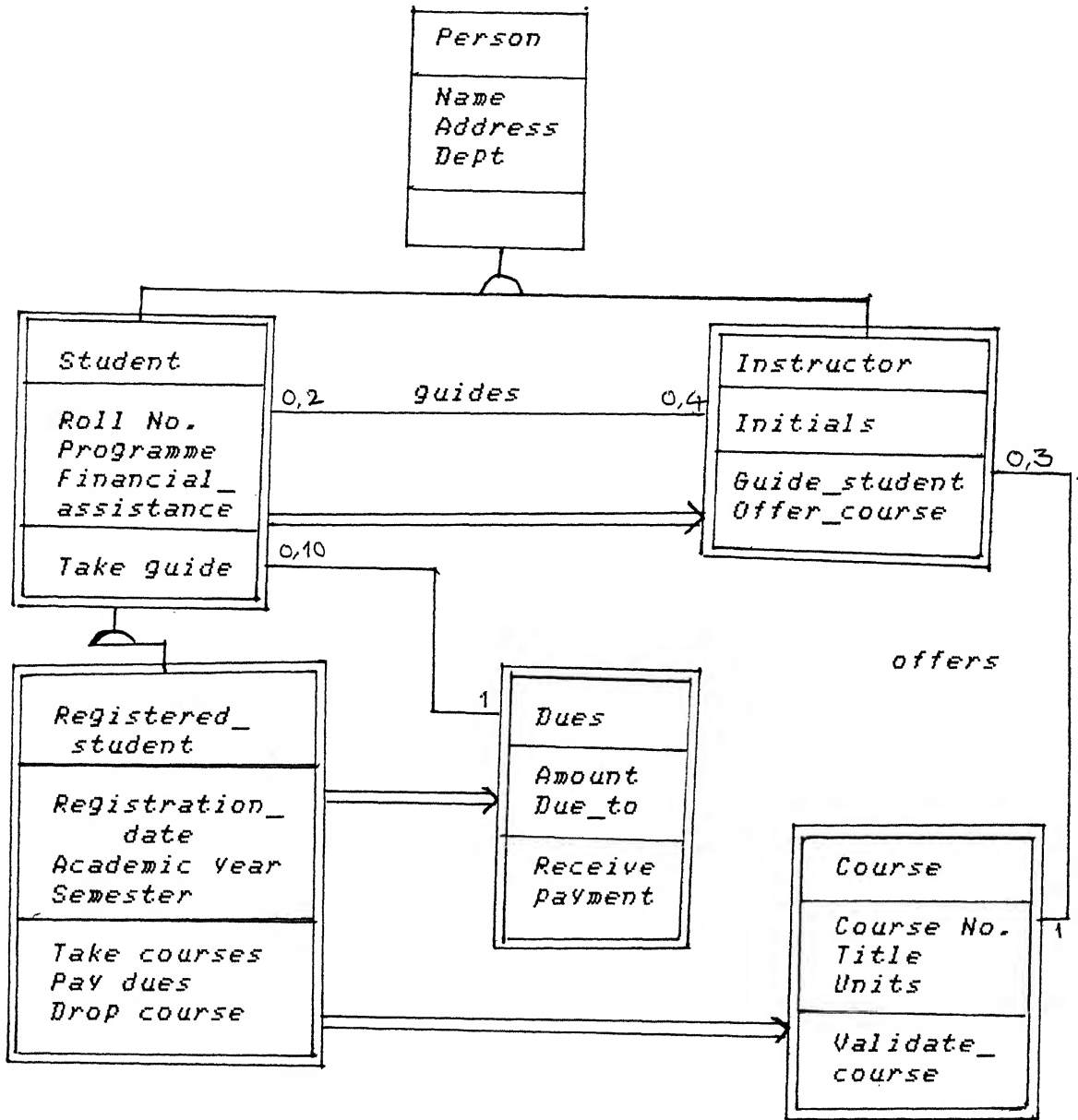


Figure 2.15: The Students Registration System - Class&Object, Structure, Attribute and Service layers

SUBJECT LAYER

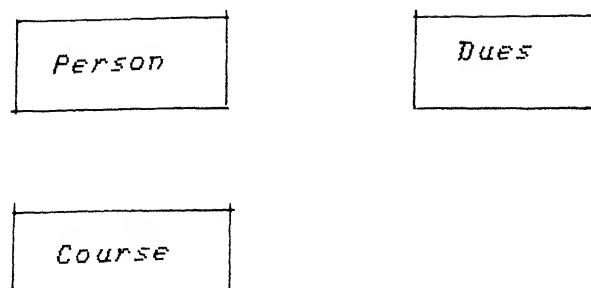


Figure 2.16a: The subjects in Students Registration System

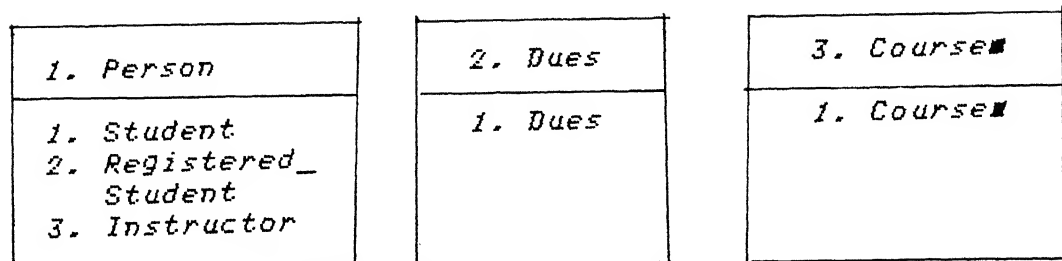


Figure 2.16b: The partially expanded subjects in Students Registration System

CLASS&OBJECT LAYER

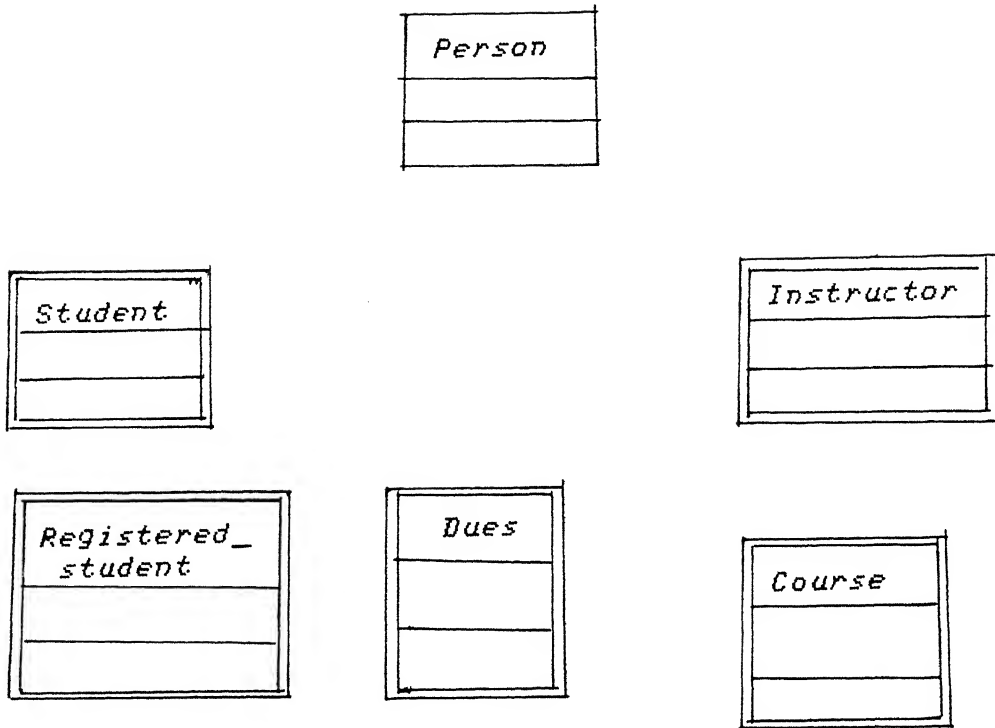


Figure 2.17: The Classes and Class&Objects in Students Registration System

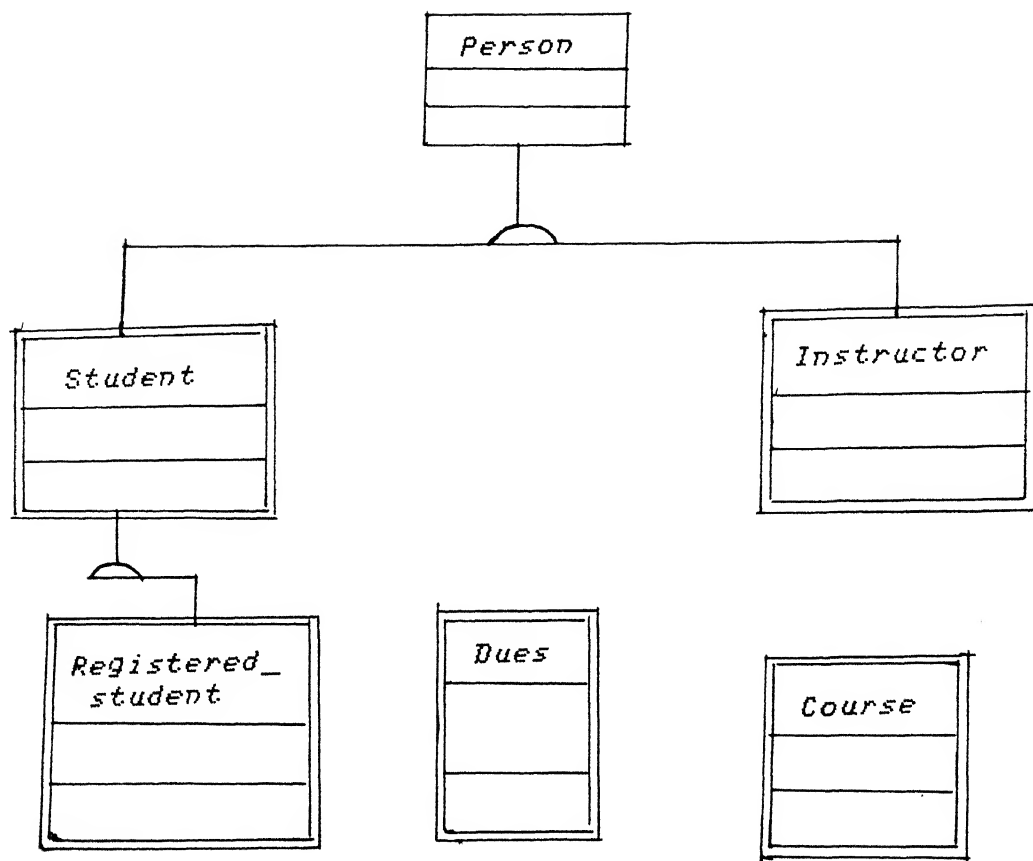


Figure 2.18: The Gen-Spec(hierarchy) structure in Students Registration System

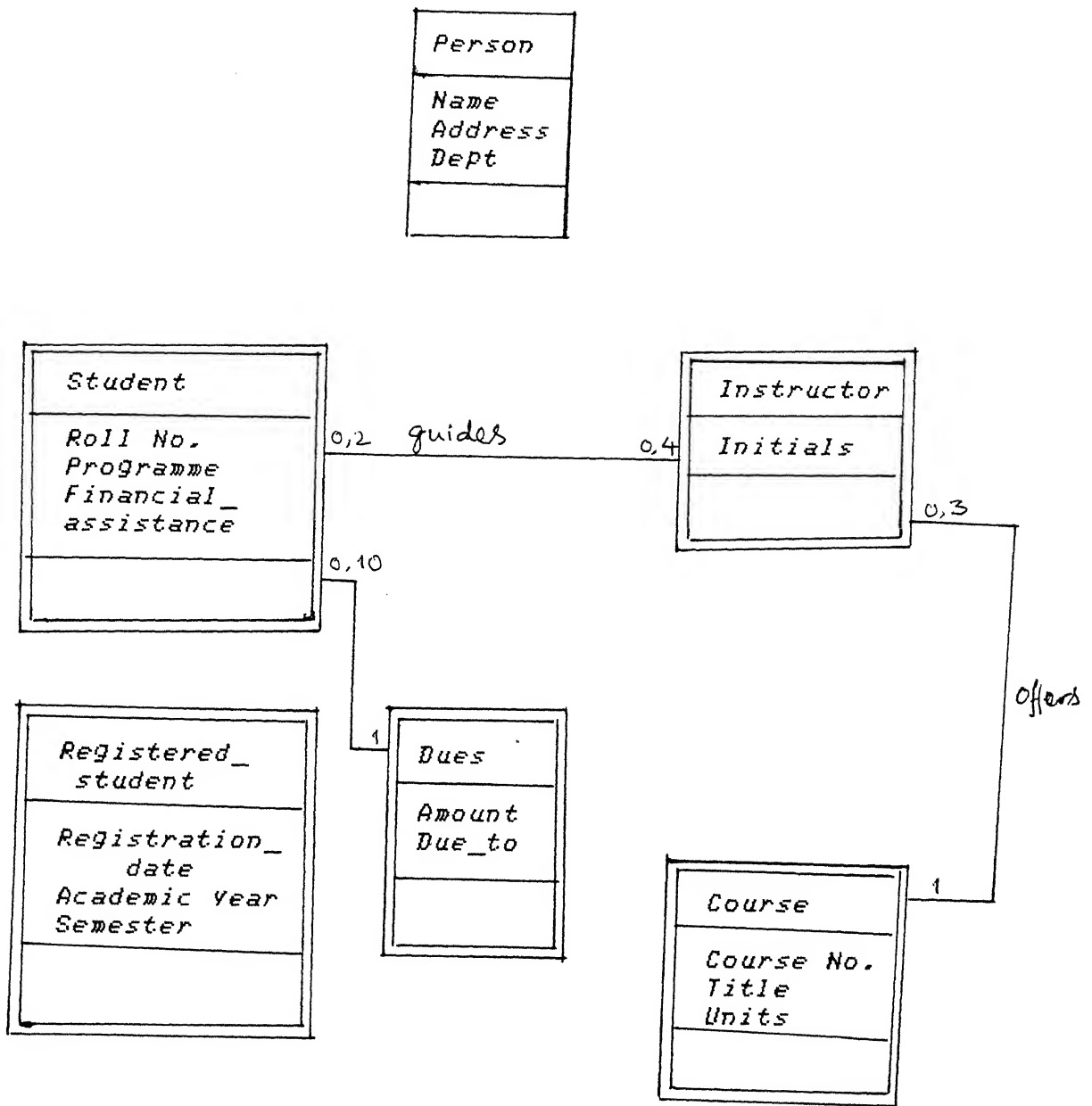


Figure 2.19: The Attribute layer for Students Registration System

SERVICE LAYER

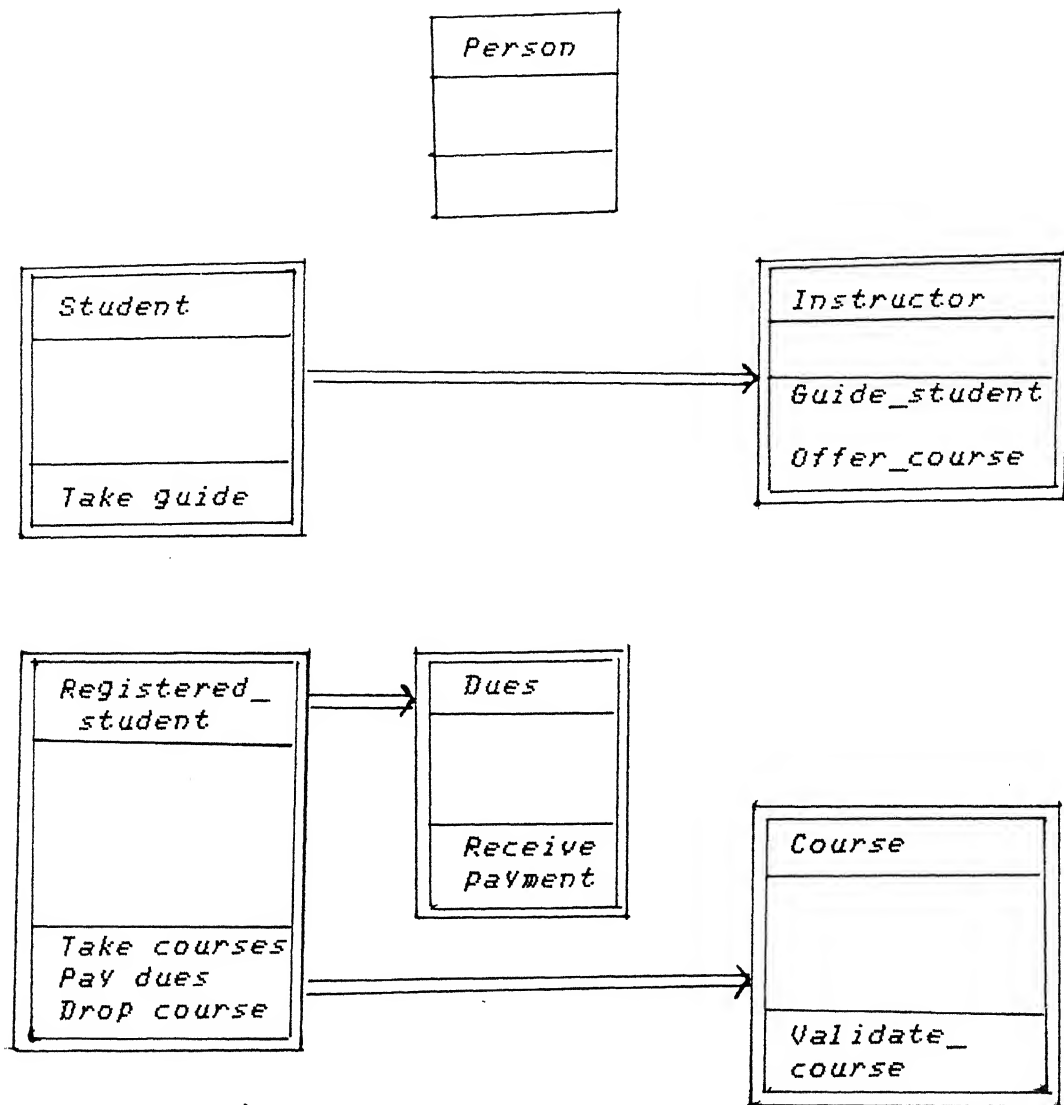


Figure 2.20: The Service layer in the Students Registration System

CHAPTER - 3

OOATool - USER INTERFACE

3.1. Introduction

A good user interface plays an important role in a software system. In future, the fast increasing raw computing power of processors will be consumed mostly in executing the user interface portion of the executable code. Designing an user interface keeping in mind both experienced user and naive user is a tough job.

In OOATool, all the interactions of the user is done through windows. The user interface contains seven types of windows. They are:

- i). Menu Window
- ii). Drawing Sheet Window
- iii). Help Window
- iv). Report Window
- v). Error Window
- vi). Dialog Window
- vii). Info Window

The screen layout of OOATool looks like:

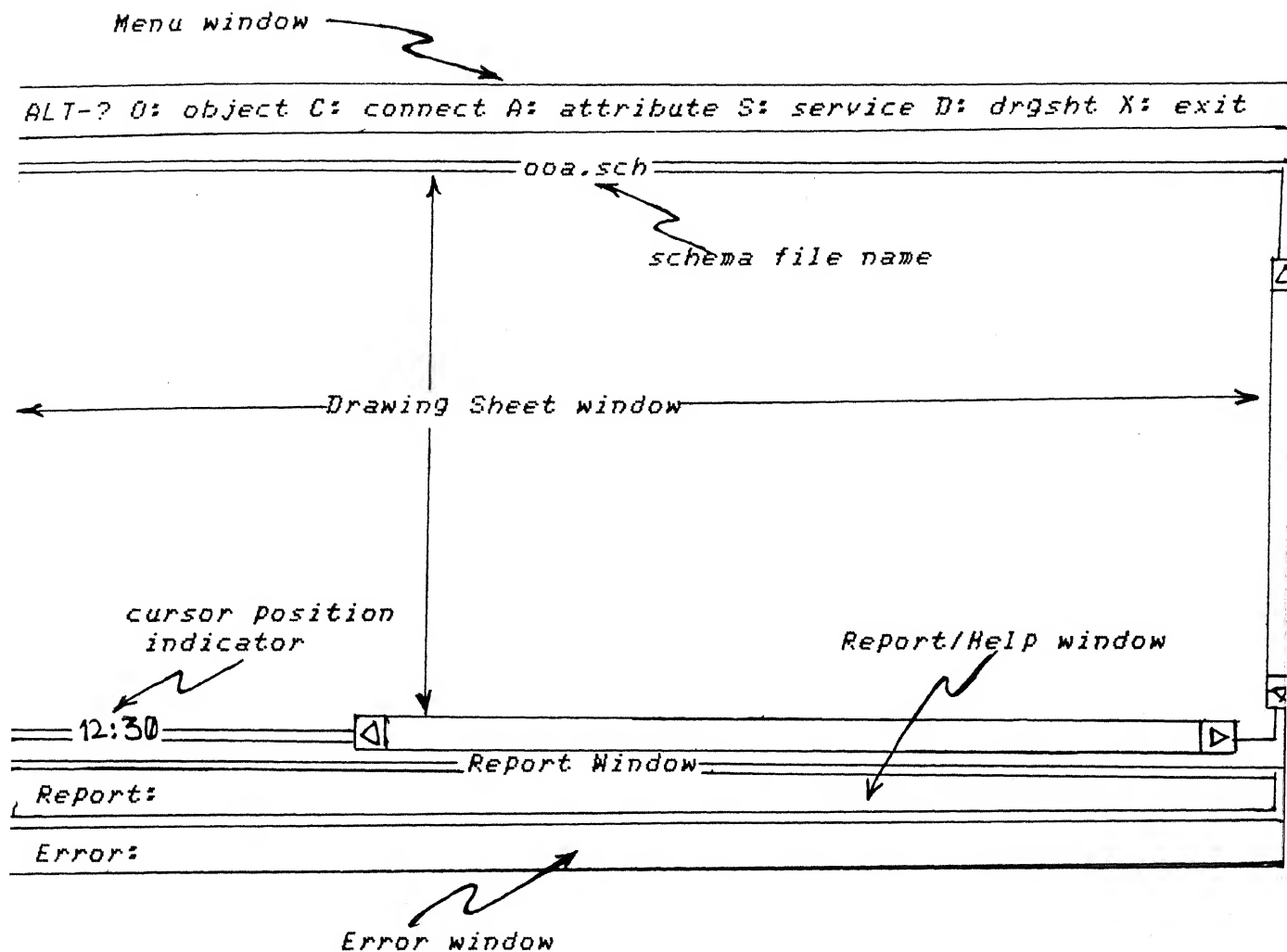


Figure 3.1: The screen layout of OOATool

We will describe each window briefly in next section.

3.2. User Interface Description

i) Menu Window :

Initially the root menu will be displayed in this win-

dow. All the commands in the root menu are ALT key followed by a character. e.g., ALT-O, ALT-C. When you press any key in the root menu, a sub-menu is displayed: The sub-menu items can be selected by just pressing corresponding item's character. In the last sub menu, when you press a key in the sub-menu, a dialog/info window is displayed at the center of the screen. The root menu and a sample sub-menu is shown below:

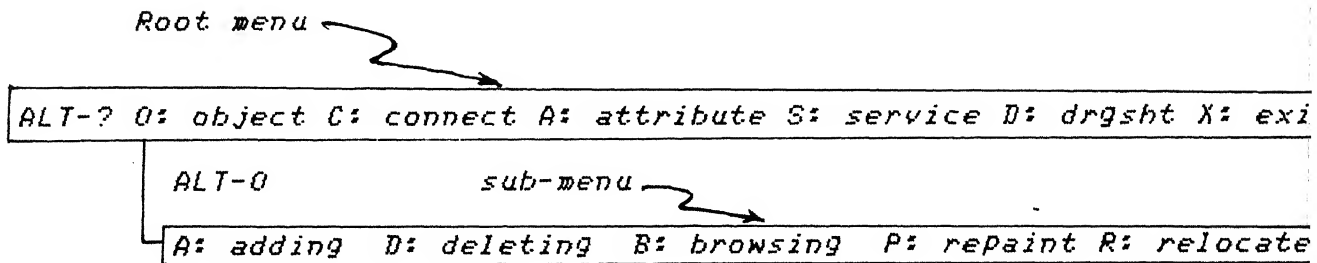


Figure 3.2: The root menu and a sample sub-menu

ii) Drawing Sheet Window :

This window shows a portion of virtual drawing sheet on which the user draws the OOA schema. The title bar, the top-most horizontal bar of the window, contains the currently opened schema file name. At the bottom left horizontal bar of the window the current cursor position is displayed as xxx:yyy.

Using arrow keys cursor can be moved in all four directions up, down, left and right. When the cursor is at the edge, moving the cursor causes the drawing sheet window

scroll in that direction. This enables to view the whole drawing sheet.

iii) Help Window :

A help message is displayed in this window, which helps the user through a sequence of operations. When a help message is displayed the system waits for some key being pressed, to ensure that the user read the message. Since the system guides the user through proper sequence of operations, he is freed from learning of some complicated sequence of schema editing operations.

iv) Report Window :

The report window overlaps with Help Window. Whenever an operation is performed on the schema, the outcome of the operation is reported in this window. Any system related messages are also displayed in this window.

e.g.: A sample report message looks like:

Report: Schema opened successfully

v) Error Window :

In the error window, errors occurred while performing an operation on schema or on drawing sheet are displayed in red color. Whenever an error is displayed, the system waits for a key press in Help Window saying:

"Did you acknowledge error message?[press any key]"

The system ensures that user has read the error message by waiting for a key press by the user.

e.g.: A sample error message looks like:

Error: The class with name: xxx not found in the schema

vi) Dialog Window :

Dialog window is designed to take input from the user before doing an operation on the schema. For example, the ALT-OIA command brings the following dialog window on the screen:

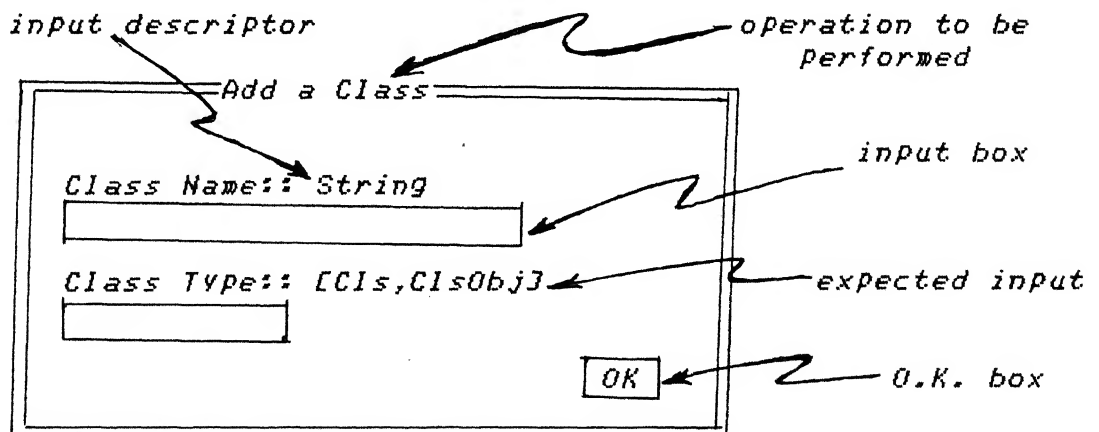


Figure 3.3: A sample dialog window of OOATool

Now the system expects the user to enter a class name(i.e., a string) and the type of the class(i.e., either Cls stands for class or ClsObj stands for Class&Object). After entering the input, select OK box and press RETURN key

there. The dialog window disappears and the corresponding operation(i.e., adding a class to the schema) is performed and the message will be displayed in Report/Error window.

vii) Info Window :

When the user wants the system to display some information, it is displayed in Info Window. For example, the ALT-OIB command brings an Info window on the screen as:

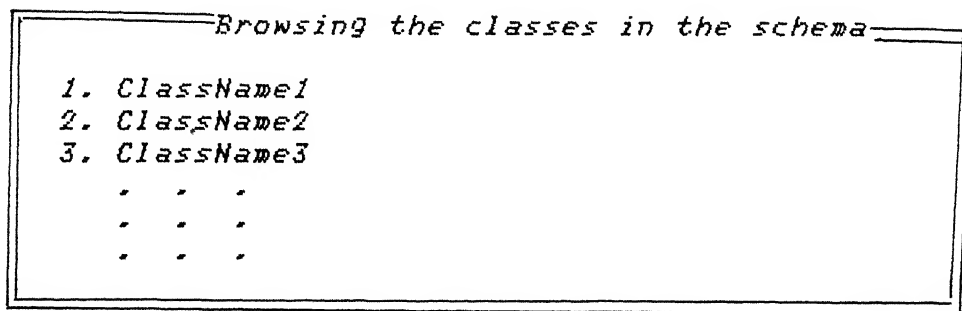


Figure 3.4: A sample info window of OOATool

The user can scroll up and down, if he wishes, in Info window. When RETURN key is pressed, a message is displayed in Help Window saying:

"Have you finished browsing through the classes?"

and waits for a key. Pressing any key will cause the Info window to disappear from the screen. Currently, OOATool allows user to browse:

a) attributes of a class

- b) services of a class
- c) instance connection of a class and
- d) whole-part connection of a class

CHAPTER - 4

DESIGN AND IMPLEMENTATION

4.1 Object-Oriented Design - An Introduction

The design of OOATool is influenced much by the Object-Oriented Design Methodology. For the sake of completeness, we briefly discuss Object-Oriented Design Methodology here.

In traditional Structured Development(i.e., the three processes of structured analysis, structured design, and structured programming), which takes functional viewpoint, functions and procedures form primary where as data is treated as secondary. But in Object-Oriented paradigm data is considered primary and procedures are secondary. The procedures(i.e., methods in Object-Oriented terms) are tied to the data.

The central problem during the Object-Oriented design process is identifying the objects and the operations defined on the objects. there are three phases in this approach:

- i) producing the initial design
- ii) function refinement
- iii) object refinement

The idea is to start with initial high-level design of

the system, with high-level objects and the functions described in terms of sub-functions. In later phases the objects and sub-functions are treated as systems and the whole process is repeated for them, until we reach a stage where the objects are "atomic objects" that can be implemented directly, and all the functions are completely defined.

i) Producing the initial design :

In this phase the objects and their operations are identified. This is done in three steps.

- a) develop a strategy to solve the problem,
- b) extract the objects and the operations on these objects from the strategy statement,
- c) establish the interface of each of the identified objects.

ii) Functional refinement :

This is an iterative process, with the input refinement coming from the initial design. As the functional refinement proceeds, new objects and operations on them are identified. In addition, during a refinement step, operation may also be identified on objects that were identified in earlier refinement steps.

When the functional refinement process terminates,

all the objects in the problem space are identified. However, more objects (and their functions) may be uncovered when object refinement takes place in the next phase. But those objects would be nested within the objects identified by the end of this phase.

iii) Object refinement :

The aim of this phase is to identify the new objects and their operations that are required in order to implement the objects. For a large system the objects may be very high-level, with complex operations that may encompass many objects.

The new objects that are identified during refinement of an object are an outcome of the attempt to specify the operations of the object undergoing refinement. Hence the purpose of these new objects is to provide services that are utilized to implement the operations of the original object undergoing refinement. These new objects should naturally be regarded as nested within the object whose refinement uncovered their existence. In such situations the nested objects are not (and need not be) visible outside the parent object.

Once all the operations on the parent object are identified, refinement of nested objects can begin. The refinement process terminates when the objects are sim-

ple enough to be implemented directly. Only for extremely large and complex systems, the nesting of objects is expected to be very deep.

For more details on the methodology the reader is requested to refer to [Jalote, 1991].

4.2. OOATool - Design and Implementation

The OOATool is implemented in C++, a block structured Object-Oriented language based on C language. The tool consists of following five modules:

- i) Modeling Manager Module
- ii) Display Manager Module
- iii) Schema Manager Module
- iv) Drawing Sheet Module
- v) Interface Manager Module

i) Modeling Manager Module:

The Modeling Manager is composed of a Display Manager and a Schema Manager. As extensions to OOATool, if we add Print Manager, Help Manager etc., all of them go will into Modeling Manager at the same level as Display Manager and Schema Manager. Modeling Manager interacts with Interface Manager to get input from the outside world and sends appropriate messages to Display Manager and/or Schema

appropriate messages to Display Manager and/or Schema Manager as required. In OOATool, the complete behavior of Modeling Manager is coded in the method `Modeling Manager::Driver()`. The C++ class implementation of Modeling Manager is listed in header file `modelm.h`, given in the Appendix.

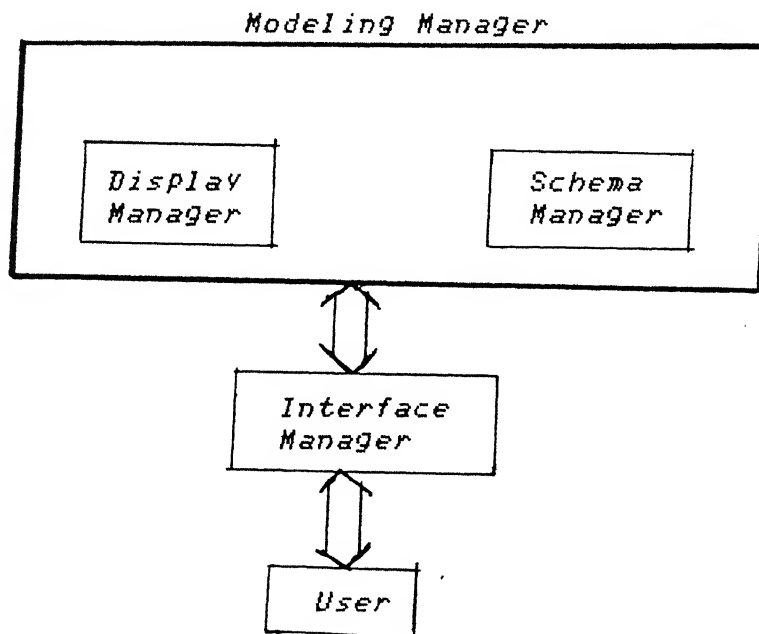


Figure 4.1: Modeling Manager Organization

ii) Display Manager Module:

The Display Manager inherits properties of Window class (the implementation of Window class is listed in the header file `window.h`, given in the Appendix. The Display Manager displays the OOA schema (i.e., drawing sheet contents) on the screen. It creates the Drawing Sheet window when a schema is loaded and performs the editing operations

on the drawing sheet when requested.

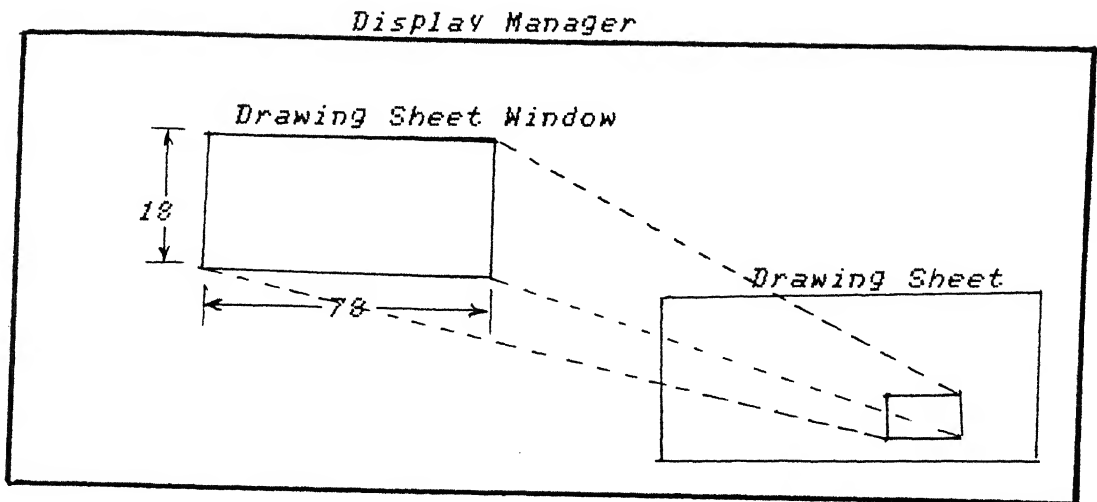


Figure 4.2: Display Manager Organization

The Display Manager contains with a drawing sheet, whose dimensions are unlimited (of course, limited by the maximum file size of the operating system). The Display Manager displays only a portion of the drawing sheet in the drawing sheet window. The user sees the drawing sheet through a 18(rows) X 78(columns) size view port. The C++ class implementation of Display Manager is listed in the header file dspm.h, given in Appendix.

iii) Schema Manager Module:

Schema Manager is responsible for managing all the classes currently entered in the schema. It maintains pointers to the classes in an array. Though the classes can be maintained as a linked list, array is preferred due to the following reasons:

- a) To avoid designing of an extra C++ class, ClassNObjectList, which is spurious and does not fit naturally in the problem space,
- b) To simplify the design and at the same time improve the readability of the design,
- c) To simplify the iteration operation over the list of classes and thus improving the performance significantly.

The information about classes in the schema is loaded(stored) from(to) a file "filename.str" when a schema is opened(closed). The C++ class implementation of Schema Manager is listed in the header file schm.h, given in Appendix.

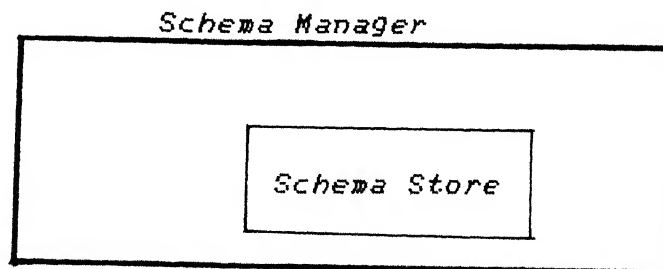


Figure 4.3: Schema Manager Organization.

iv) Drawing Sheet Module:

This module implements a virtual two-dimensional drawing sheet through a UNIX FILE stream. It provides facilities to write a string horizontally or vertically on the drawing sheet and to read a chunk of byte from FILE stream. The C++

class implementation of Drawing Sheet is listed in the header file drgsh.h, given in Appendix.

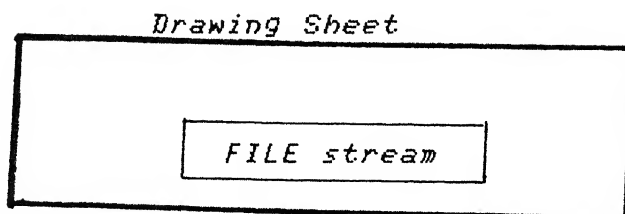


Figure 4.4: Drawing Sheet Organization.

v) Interface Manager Module:

This module is a set of classes; each class is a subclass of Window class. When an instance of one of these classes is created, its interface window is displayed on the screen. It either takes input from the user or displays schema related information in the displayed window. The window disappears from the screen as soon as the instance is destroyed. These classes are utilized by the Modeling Manager module when it wants to interact with the user. The C++ class implementation of Interface Manager is listed in the header files intfwin.h, intfwins.h and infowin.h in Appendix.

Finally, we describe a simple and uniform communication protocol that is followed among the various modules of OOA-Tool. The modules communicate with each other through a message, which has been implemented as C++ Message class. Whenever a sender(a method) sends a message to receiver(a method in the same class or in another class) the receiver

replies the outcome of message execution by creating an instance of Message and returning it to the sender. The sender after receiving the Message from receiver does the required clean-up operations based on the outcome. The C++ class implementation of Message is listed in the header file message.h given in Appendix.

CHAPTER - 5

CONCLUSIONS AND FURTHER WORK

If the system under consideration is large, the number of objects identified will also be large. Then it becomes difficult for the analyst to remember/recollect the names of the all classes, its attributes, its services and other information. The analysis document for such large system tend to become so huge that ensuring consistency and correctness when a slight modification is done to a portion of the schema will be a tedious and painstaking job. Under such circumstances, using a tool like OOATool will definitely increase the productivity of the analyst.

Even though OOATool provides editing and document generation facilities, present implementation does not meet many demanding features for a professional analyst. We foresee the following extensions to OOATool.

- i) facility to categorize attributes and services as private, protected and public, which enables controlling the access to them by sub-class and other classes(in the spirit of C++),
- ii) support for specifying the all services that are needed by a service of a class,
- iii) adding support for subject layer, which is

presently missing in this implementation,

- iv) on-line help facility giving directions how to go about analyzing further at any stage,
- v) provisions for editing multiple OOA schema simultaneously,
- vi) presently the number of classes that can be entered in the schema are limited by the operating system limitations (like in DOS on personal computers, maximum memory is 640KB). This is because present implementation stores all schema related data in main memory. To support large number of classes OOATool can be linked to a Database Management System to store schema related information.

References

1. [Chen, 1976] Chen, P. P., *The Entity-Relationship model - Toward a unified view of data*, ACM Trans. Database Syst. Vol. 1, No. 1, Mar. 1991.
2. [Coad and Yourdon, 1991a] Coad, Peter and Yourdon, Edward, *Object-Oriented Analysis*, 2/e, Yourdon Press, Prentice Hall, 1991.
3. [DeMacro, 1978] Demacro, Tom, *Structured Analysis and Systems Specification*, Prentice Hall, 1978.
4. [Flavin, 1981] Flavin, Matt, *Fundamental Concepts of Information Modeling*, Prentice Hall, 1990.
5. [Gane and Sarson, 1979] Gane, Chris and Sarson, Trish, *Structured Systems Analysis: Tools and Techniques*, Prentice Hall, 1979.
6. [Hammer and McLeod, 1981] Hammer, M., and McLeod, D., *Database description with SDM: A semantic database model*, ACM Trans. Database Syst., Vol. 6, No. 3, Sept. 1981.
7. [Jalote, 1991] Jalote, Pankaj, *An Integrated Approach to Software Engineering*, Narosa Publishing House (also Springer-Verlag), 1991.
8. [McMenamin and Palmer, 1984] McMenamin, Steve and Palmer, John, *Essential systems Analysis*, Prentice Hall, 1984.
9. [Patrick, 1990] Patrick, H. Loy, *Comparisons of Object-Oriented and Structured development methods*, ACM SIGSOFT, Software Engineering Notes, Vol.15, No. 1, Jan. 1990.

10. [Shlaer and Mellor, 1988] Shlaer, Sally and Mellor, Steve. *Object-Oriented Systems Analysis*, Prentice Hall, 1988.
11. [Teorey et al., 1986] Teorey, T. J., Yang D., and Fry, J. P., *A logical design methodology for relational databases using the extended entity-relationship model*, ACM Comput. Surv., Vol. 18, No. 2, June 1986.
12. [Yourdon and Constantine, 1979] Yourdon, Edward, Constantine, L., *Structured Design*, Prentice Hall, 1979.
13. [Yourdon, 1989] Yourdon, Edward, *Modern Structured Analysis*, Prentice Hall, 1989.

CENTRAL LIBRARY
UNIVERSITY OF KANSAS

Acc. No. A114058

Bibliography

1. [Borland, 1990a] Borland International Inc., *Turbo C++ : Getting Started*, 1990.
2. [Borland, 1990b] Borland International Inc., *Turbo C++ : User's Guide*, 1990.
3. [Borland, 1990c] Borland International Inc., *Turbo C++ : Programmer's Guide*, 1990.
4. [Coad and Yourdon, 1991b] Coad, Peter and Yourdon, Edward, *Object-Oriented Design*, Prentice-Hall, 1991.
5. [Manola and Dayal, 1986] Manola, F., and Dayal, U., *PDM: An Object-Oriented data model*, In Proceedings of the Workshop on Object-Oriented Databases(Pacific Grove, Calif.), IEEE, New York, Sept. 1986.
6. [Meyer, 1990], Meyer, B., *Lessons from the Design of the Eiffel Libraries*, CACM, Vol. 33, No. 9, Sept. 1990.
7. [Stroustrup, 1986], Stroustrup, Bjarne, *The C++ Programming Language*, Addison-Wesley, Reading, Mass. 1986.
8. [Stroustrup, 1988], Stroustrup, Bjarne, *What is Object-Oriented Programming*, IEEE Software, May, 1988.

Appendix-A

User's Manual

Hardware and Software requirements:

OOATool runs on the IBM PC family of computers. It requires DOS 2.0 or higher and with at least 640KB. It runs on any 80-column monitor. The minimum requirement is a hard disk drive and one floppy drive. OOATool is to be run from hard disk drive for faster response time. OOATool accesses the schema file "xxx.sch" most frequently since whenever the drawing sheet is scrolled in the drawing sheet window, the contents are read from the file "xxx.sch". OOATool is implemented in Turbo C++ version 1.00[Borland, 1990a], [Borland, 1990b],[Borland, 1990c]. To start OOATool type:

```
C:> ootool <CR>
```

The tool first displays a welcome message and opens a default schema with file name "ooa.sch".

Loading and Closing a schema:

If you want to load a schema other than the default "ooa.sch", press <F3> key. Now the system closes the current schema and asks for a new schema file name. The file name "xxx", should be entered without the extension ".sch". The system will open the schema file as "xxx.sch". The current schema can be closed by <ALT-F3> key.

Menu Reference:

The following command keys are available in OOATool.

ALT-O : for doing operations related to Class.

ALT-C : for doing operations related to Connection (Gen-Spec/Whole- Part structure or Instance/Message connections)

ALT-A : for doing operations related to attribute of a Class

ALT-S : for doing operations related to service of a class

ALT-D : for drawing sheet related operations

ALT-X : exiting from OOATool.

ALT-O menu:

A: adding :-

This command can be selected by pressing 'a' or 'A' character. The command displays a dialog window expecting "Class Name" and "Class Type" from the user. You can move over the input boxes by up and down keys or TAB key. After entering the input, select OK box and press return.

D: deleting :-

This command deletes the names class from the schema.

B: browsing :-

You can choose this command when you want to see all the Classes that are there in the schema.

P: repaint :-

This command is chosen when you want to

repair(repaint) an Class or its connection. Choosing this command redraws the Class and all its connections and thus repairing any damage.

R: relocate:-

You choose this command when you want to make space on the drawing sheet by relocation(moving) the object to different location on the drawing sheet.

ALT-C menu:

G: Gen-Spec :-

This command displays a sub-menu consisting of Gen-Spec connections types: H: hierarchy connection
L: lattice connection

Choosing any one option will display another sub-menu with the following two options:

A: adding :-

adds the specified Gen-Spec connection.

D: deleting :-

deletes the selected Gen-Spec connection.

W: Whole-Part :-

This command displays a sub-menu with items

A: adding :-

adds the specified Whole-Part connection.

D: deleting :-

deletes the selected Whole-Part connection.

B: Browsing :-

displays the complete whole-part connection in an info window.

I: Instance :-

This command displays a sub-menu identical to the sub-menu of Whole-part Menu.

M: Message :-

This command displays a sub-menu with the items

A: Adding :-

adds a specified message connection

D: Deleting :-

deletes a selected message connection

ALT-A menu:

ALT-S menu: These two commands display a sub-menu with items

A: adding :-

adds an attribute/service to an object.

D: deleting :-

deletes an attribute/service from an object.

B: browsing :-

displays all the attributes/services of an object in an info window.

ALT-D menu:

E: use eraser :-

makes the cursor as eraser. This is useful when some stray characters on the drawing sheet are to be erased.

X: expand :-

expand the size of the drawing sheet to specified dimensions.

T: generate templates :-

generates C++ templates in a file xxx.tmp.

D: generate document :-

generates an analysis document for the schema in a file xxx.doc.

<F2> Key : Stores the schema to a file xxx.str.

<F8> Key : displays the available main memory. This gives you an idea how safe it is to do an operation without running out of main memory. This prevents the systems from hanging up if it runs out of memory.

<Home> Key, <End> Key : These keys are context sensitive. The function of the key in a context is reported in the help window.

<Up Arrow>, <Down Arrow>, <Left Arrow> and <Right arrow> Keys : These keys move the cursor on the drawing sheet window.

<Shift-Home>, <Shift-End>, <Shift-PgUp>, <Shift-PgDn> Keys : These keys move the drawing sheet left, right, up and down by the full dimension of the drawing sheet window(i.e., shifts by 70 columns horizontally or shifts by 8 rows vertically).

<Ctrl-Home>, <Ctrl-End>, <Ctrl-PgUp>, <Ctrl-PgDn> Keys: These keys move the drawing sheet left, right, up and down by half the dimension of the drawing sheet window(i.e., shifts by 35 columns horizontally or shifts by 7 rows vertically).

OOA Notation in OOATool:

The exact notation of OOA is not followed due to implementation problems (the problem is OOA TOOL uses the screen in Text mode and one cannot display graphic notation in text mode). An alternate notation is followed. Class and Class&Object:

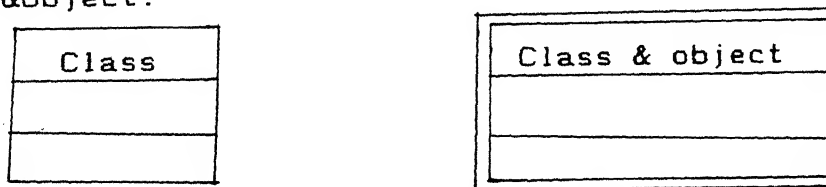


Figure A.1: The OOATool "Class" and "Class&Object" symbols

Gen-Spec Hierarchy Connection (Structure):

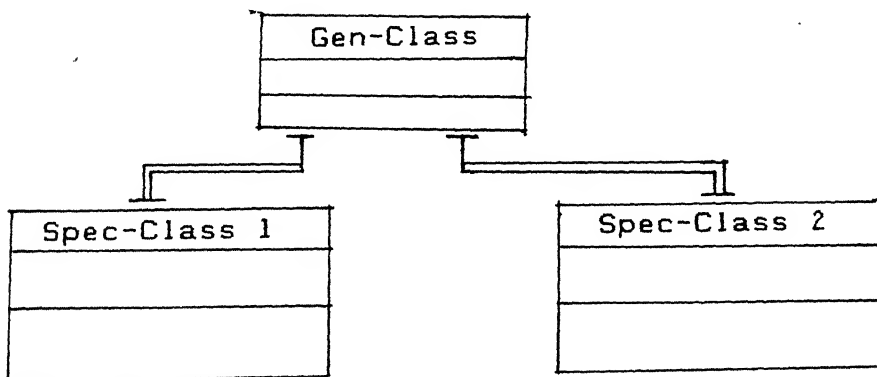


Figure A.2a: The OOATool "Gen-Spec Hierarchy" Structure notation

Gen-Spec-Lattice Connection(Structure):

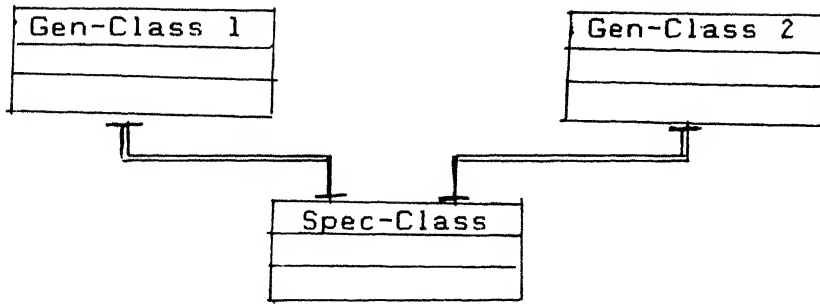


Figure A.2b: The OoATool "Gen-Spec Lattice" Structure notation

A Gen-Spec Connection should exist only between a Class and another Class [Coad & Yourdon, 1991].

Instance Connection:

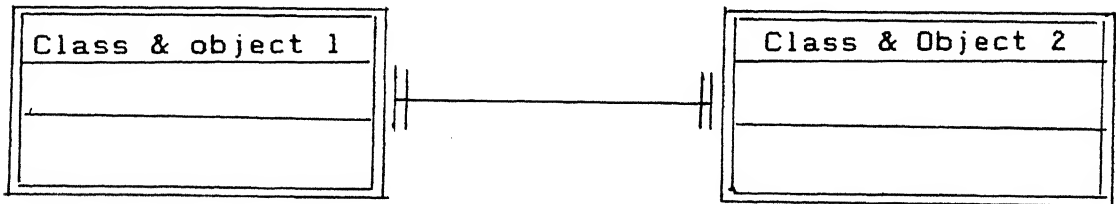


Figure A.3: The OoATool "Instance" Connection notation

Instance connection should exist only between a Class&Object to another Class&Object.

Message Connection:

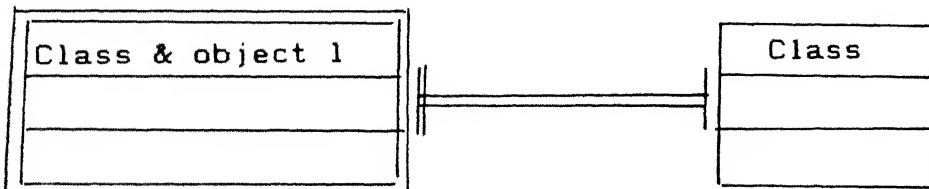
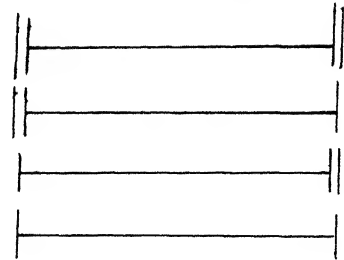


Figure A.4: The OoATool "Message" Connection notation

A message connection can be of the following four types

1. Object to Object
2. Object to Class
3. Class to Object
4. Class to Class

Notation



In ODATOOL a class is always displayed as:

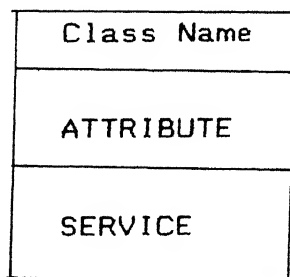


Figure A.5: Display of a class in ODATool

The attribute and services of a class are not displayed on their corresponding horizontal sections. This is due to following two reasons:

1. User will not be interested in looking at the attributes and services always.
2. To minimize the screen space occupied by a class symbol on the drawing sheet window, so that more classes can be displayed on the screen at any time.

The following files are generated by ODATool. Let the name of the schema opened be "xxx".

File Name	Description
xxx.sch	This file contains the drawing sheet. The schema diagram that is shown in the drawing sheet window is stored in this file.
xxx.str	This file stores all information about the classes in the schema.
xxx.inf	This file stores extra information about the attributes and services of a class.
xxx.tmp	The C++ class templates for the classes in the schema are stored in this file.
xxx.doc	The analysis document that is generated for the schema is stored in this file.


```

        AddWholePartConnect( void ),
        AddInstanceConnect( void ),
        AddMessageConnect( void );

void    DelGenSpecHyConnect( void ),
        DelGenSpecLatConnect( void ),
        DelWholePartConnect( void ),
        DelInstanceConnect( void ),
        DelMessageConnect( void );

void    DisplayClasses( void ),
        DisplayWholePartConnect( void ),
        DisplayInstanceConnect( void ),
        DisplayAttributes( void ),
        DisplayServices( void );

void    Driver( void ),
        RedrawObjectAndConnects( char * ),
        ResetCursor( void );

Message *RelocateObjectAndConnects( char *, Location * );

};

inline ModelingManager::ModelingManager( char *fname )
{
    OpenSchema( fname );
}

inline ModelingManager::~ModelingManager()
{
    CloseSchema();
}

inline void ModelingManager::GetDrgShtLoc( Location *loc )
{
    DspMgr->GetLoc( loc );
}

inline KEY ModelingManager::RoamOnScreen()
{
    return DspMgr->RoamOnScreen();
}

inline void ModelingManager::ResetCursor( void )
{
    DspMgr->ResetCursor();
}

#endif

```

```

/***** Class implementation of *****/
***** Display Manager *****/

```

```

#ifndef _DSPM_H
#define _DSPM_H

#include "drgsh.h"
#include "window.h"
#include "message.h"

const SCREENWIDTH = 80, //no of columns in the CRT screen Text mode
      SCREENHEIGHT = 25; //no of rows in the CRT screen in Text mode

const MAX_DISP_ATTR = 1,
      MAX_DISP_SERV = 1;

const BOXLEN = 17,
      BOXWID = 5+MAX_DISP_ATTR+MAX_DISP_SERV;

const FULLSCREEN = SCREENHEIGHT -4;

class DisplayManager : private Window {
private:
    DrawingSheet *DrgSht;
    unsigned      DLength,
                  DHeight;

    unsigned      *DspLnPtrs; /* file offsets in the drawing sheet
                               of the lines that are currently
                               displayed in the drawing sheet window
                               */
    char          *DspBuf;    /* display buffer for the drawing sheet
                               window */

    void          BufLeftScroll( unsigned ),
                  BufRightScroll( unsigned ),
                  BufUpScroll( unsigned ),
                  BufDnScroll( unsigned );

    void          MoveCurLeft( unsigned ),
                  MoveCurRight( unsigned ),
                  MoveCurUp( unsigned ),
                  MoveCurDown( unsigned );

    Message *GetLinePoints( Location ** );
    void      DrawLine( Location *, Location *, char, char ),
             DrawPolyLine( Location **, char, char, char, char ),
             PutChar( char, Location * );

```

public:

65

```
    DisplayManager( char *fname ="ooo.sch",
                    int left =1, int top =2,
                    int right =SCREENWIDTH,
                    int btm =SCREENHEIGHT-4,
                    int fg =YELLOW, int bg =BLUE);
    ~DisplayManager();

    void    OutLine( void );

    void    ResetCursor( void ) { ResetCur(); }

    void Show( void ),
        OpenDrgSht(char *),
        ExpandDrgSht( int, int ),
        DrawObject( char *, ClassType, Location * ),
        EraseObject( Location * ),
        UseEraser( void );

    void    DrawMessageConnect( Location *, Location *, Location **,
                               ConnectionCategory ),
        DrawInstanceConnect( Location *, Location *, Location ** ),
        DrawWholePartConnect( Location *, Location *, Location ** ),

        DrawGenSpecHyConnect( Location *, Location *, Location ** ),
        DrawGenSpecLatConnect( Location *, Location *, Location ** );

    Message *MoveVerticalConnect( Location *, Location *, Location *,
                                  Location *, Location *, Location * ),
        *MoveHorizontalConnect( Location *, Location *, Location *,
                                 Location *, Location *, Location * ),

        *ShiftGenSpecHyConnect( Location *, Location *, Location ** ),
        *ShiftGenSpecLatConnect( Location *, Location *, Location ** ),
        *ShiftWholePartConnect( Location *, Location *, Location ** ),

        *ShiftInstanceConnect( Location *, Location *, Location ** ),
        *ShiftMessageConnect( Location *, Location *, Location ** );

    void    RedrawGenSpecHyConnect( Location ** ),
        RedrawGenSpecLatConnect( Location ** ),
        RedrawWholePartConnect( Location ** ),
        RedrawInstanceConnect( Location ** ),
        RedrawMessageConnect( Location **, ConnectionCategory );
```

```

void    ErasePolyLine( Location ** );
void    GetLoc( Location * );

int     RoamOnScreen( void );
void    LeftMove( unsigned ),
        RightMove( unsigned ),
        UpMove( unsigned ),
        DownMove( unsigned );

int     RetDrgShtLength() { return DrgSht->Length(); }
int     RetDrgShtWidth() { return DrgSht->Width(); }

};

#endif

```

```

/***** Class implementation of *****/
***** Schema Manager *****/

#ifndef _SCHM_H
#define _SCHM_H

#include "clsobj.h"
#include "common.h"
#include "message.h"
#include "ooa.h"

#include <stdio.h>

const MAX_OBJECTS = 50;

class SchemaManager {
private:
    unsigned    ObjCount;
    ClassNObject *SchList[ MAX_OBJECTS ];

    char        *Fname;
    FILE        *SchStore;

    ClassNObject *RetObjPtr( char * ),
                *RetObjPtr( ObjectId );

    void    DelGenSpecHyConnect( GenSpecConnection * ),
            DelGenSpecLatConnect( GenSpecConnection * ),
            DelWholePartConnect( WholePartConnection * ),
            DelInstanceConnect( InstanceConnection * ),
            DelMessageConnect( MessageConnection * );

public:
    SchemaManager( char *="ooa.str" );
    ~SchemaManager();

    friend class ModelingManager;

    void    LoadSchema( void ),
            StoreSchema( void ),
            LoadDocumentInfo( void ),
            StoreDocumentInfo( void ),

            GenerateTemplates( void ),
            GenerateDocument( void );

    Message *AddClsNObj( char *, ClassType, Location * ),
            *DelClsNObj( char * );

    Message *AddGenSpecHyConnect( char *, char * ),
            *DelGenSpecHyConnect( char *, char * ),

```

```

*AddGenSpecLatConnect( char *, char * ),
*DelGenSpecLatConnect( char *, char * ),

*AddWholePartConnect( char *, char *,
                      int =Constraint::NOT_APPLY,
                      int =Constraint::NOT_APPLY,
                      int =Constraint::NOT_APPLY,
                      int =Constraint::NOT_APPLY ),
*DelWholePartConnect( char *, char * );

Message *AddInstanceConnect( char *, char *,
                             int =Constraint::NOT_APPLY,
                             int =Constraint::NOT_APPLY,
                             int =Constraint::NOT_APPLY,
                             int =Constraint::NOT_APPLY ),
*DelInstanceConnect( char *, char * ),

*AddMessageConnect( char *, char *, ConnectionCategory ),
*DelMessageConnect( char *, char *, ConnectionCategory );

Message *AddAttribute( char *objname, char *attrname),
*DelAttribute( char *objname, char *attrname ),
*AddAttrTypeAndSeman( char *objname, char *attrname,
                      char *type, char *seman ),

*AddService( char *objname, char *sename ),
*DelService( char *objname, char *sename ),
*AddServTypeAndSeman( char *objname, char *servname,
                      char *type, char *seman);

Location  **RetGsHierarchyPtsList( char *, char * ),
          **RetGsLatticePtsList( char *, char * ),
          **RetWpConPtsList( char *, char * ),
          **RetInstConPtsList( char *, char * ),
          **RetMsgConPtsList( char *, char *, ConnectionCategory );

ObjectId  RetOid( char *objname );
void      RetObjLoc( char *, Location * ),
          GetClassName( ObjectId, char * );

Message *GetClassList( char ** ),
*GetWholePartConInfo( char *, char *, int *, int *,
                      int *, int * ),
*GetInstanceConInfo( char *, char *, int *, int *,
                      int *, int * ),
*GetAttributeList( char *, char ** ),
*GetServiceList( char *, char ** );

void      SetAttrTypeAndSeman( char *, char *, char *, char * ),
          SetServTypeAndSeman( char *, char *, char *, char * );
};

#endif

```

```

/***** Class implementation of *****/
***** Drawing Sheet *****/

#ifndef __DRGSH_H
#define __DRGSH_H

#include "ooa.h"

#include <stdio.h>

const DRGLEN = 100, // default length of the Drawing Sheet.
      DRGWID = 30;  // default width of the Drawing Sheet.

class DrawingSheet {
private:
    unsigned    Len,
                Wid;

    char        *Fname;
    FILE        *SchemaFile; /* stores the virtual drawing sheet */

    void        OpenSchemaFile( char * );

public:

    DrawingSheet( char * ="ooa.sch",
                  unsigned =DRGLEN, unsigned =DRGWID );
    ~DrawingSheet();

    unsigned    Length();
    unsigned    Width();

    /* the following three methods are used to read from FILE stream */
    void        Seek(long offset, int base);
    void        Read(char *buf, int count);
    long        Tell();

    void        Open( char * ),
                Close( void ),
                Expand( int, int ),
                WriteHString( char *, Location * ),
                WriteVString( char *, Location * );

};

#endif

```

```

/***** Class implementation of *****/
***** Class&Object *****/

#ifndef _CLSNBJ_H
#define _CLSNBJ_H

#include "connects.h"
#include "attrserv.h"
#include "ooa.h"
#include "message.h"

const    MAX_GS_CONNECTIONS = 15, MAX_WP_CONNECTIONS = 15,
          MAX_MSG_CONNECTIONS = 15, MAX_INST_CONNECTIONS = 15,
          MAX_ATTRIBUTES = 25,    MAX_SERVICES = 25;

enum DeleteType {
    NOT_DESTROY = 0, DESTROY = 1
};

class ClassObject {
private:
    ObjectId      Oid;
    char          *ClsName;
    ClassType     Type;

    GenSpecConnection *GsHierarchy[MAX_GS_CONNECTIONS],
                    *GsLattice[MAX_GS_CONNECTIONS];
    MessageConnection *MsgCons[MAX_MSG_CONNECTIONS];
    WholePartConnection *WpCons[MAX_WP_CONNECTIONS];
    InstanceConnection *InstCons[MAX_INST_CONNECTIONS];

    Attribute       *Attributes[MAX_ATTRIBUTES];
    Service          *Services[MAX_SERVICES];

    Location         *Loc;

    static unsigned  NextOid;

    // the following methods are called by
    // Schema Manager while loading schema
    GenSpecConnection *RetGsHierarchyPtr( ObjectId, ObjectId ),
                    *RetGsLatticePtr( ObjectId, ObjectId );
    WholePartConnection *RetWpConPtr( ObjectId, ObjectId );
    InstanceConnection *RetInstConPtr( ObjectId, ObjectId );
    MessageConnection *RetMsgConPtr( ObjectId, ObjectId,
                                    ConnectionCategory );

public:
    ClassObject( char * =NULL, ClassType =CLASS,
                Location * =NULL );
    ~ClassObject();

```



```

friend class SchemaManager;
friend class ModelingManager;

static unsigned GenOid() { return NextOid++; }
static void      SetOid( unsigned seed ) { NextOid = seed; }

Message  *AddGenSpecHyConnect( GenSpecConnection * ),
          *DelGenSpecHyConnect( GenSpecConnection *,
                                DeleteType =DESTROY ),

          *AddGenSpecLatConnect( GenSpecConnection * ),
          *DelGenSpecLatConnect( GenSpecConnection *,
                                DeleteType =DESTROY ),

          *AddWholePartConnect( WholePartConnection * ),
          *DelWholePartConnect( WholePartConnection *,
                                DeleteType =DESTROY ),

          *AddInstanceConnect( InstanceConnection * ),
          *DelInstanceConnect( InstanceConnection *,
                                DeleteType =DESTROY ),

          *AddMessageConnect( MessageConnection * ),
          *DelMessageConnect( MessageConnection *,
                                DeleteType =DESTROY ),

          *AddAttribute( Attribute * ),
          *DelAttribute( char * ),
          *AddAttrTypeAndSeman( char *, char *, char * ),

          *AddService( Service * ),
          *DelService( char * ),
          *AddServTypeAndSeman( char *, char *, char * );

Location **RetGsHierarchyPtsList( Objectid, Objectid ),
          **RetGsLatticePtsList( Objectid, Objectid ),
          **RetWpConPtsList( Objectid, Objectid ),
          **RetInstConPtsList( Objectid, Objectid ),
          **RetMsgConPtsList( Objectid, Objectid,
                                ConnectionCategory );

Objectid RetOid( void ) { return Oid; }
char *   RetClsName( char *buf )
        { return strcpy( buf, ClsName ); }
ClassType RetClsType( void ) { return Type; }
Location  *RetLoc( void ) { return Loc; }
};

#endif

```

```

/***** Class implementation of *****/
/***** Structures and Connections *****/

#ifndef __CONNECTS_H
#define __CONNECTS_H

#include "boa.h"

class Connection {
protected:
    ObjectId    FromDid, ToDid;

public:
    Location     *ConPtsList[ MAX_CONN_PTS ]; // public member field

    Connection( ObjectId, ObjectId );
    ~Connection();

    Location     **RetPtsList( void );

    ObjectId     RetFromDid( void );
    ObjectId     RetToDid( void );
};

inline Location **Connection::RetPtsList()
{
    return ConPtsList;
}

#endif

#ifndef __CONNECT_H
#define __CONNECT_H

#ifndef __CONSTRAINT
#define __CONSTRAINT

class Constraint {
    int Val1, Val2;
    enum { NOT_APPLY = -1 };

public:
    Constraint( int v1 =NOT_APPLY, int v2 =NOT_APPLY ) {
        Val1 = v1; Val2 = v2;
    }

    void SetVal1( int v ) { Val1 = v; }
    void SetVal2( int v ) { Val2 = v; }

    int RetVal1() { return Val1; }
    int RetVal2() { return Val2; }
};

```

```

#endif

class ConstrainedConnection : public Connection {
protected:
    Constraint    FromConst, ToConst;

public:
    ConstrainedConnection( ObjectId, ObjectId,
                          int fval1 =Constraint::NOT_APPLY,
                          int fval2 =Constraint::NOT_APPLY,
                          int tval1 =Constraint::NOT_APPLY,
                          int tval2 =Constraint::NOT_APPLY );

    int RetFromConstVal1();
    int RetFromConstVal2();
    int RetToConstVal1();
    int RetToConstVal2();

};

#endif

#ifndef __WPCON_H
#define __WPCON_H

class WholePartConnection : public ConstrainedConnection {
    // no member fields
public:
    WholePartConnection( ObjectId, ObjectId,
                      int fval1 =Constraint::NOT_APPLY,
                      int fval2 =Constraint::NOT_APPLY,
                      int tval1 =Constraint::NOT_APPLY,
                      int tval2 =Constraint::NOT_APPLY );

};

#endif

#ifndef __GSCON_H
#define __GSCON_H

class GenSpecConnection : public Connection {
    // no member fields
public:
    GenSpecConnection( ObjectId, ObjectId );

};

#endif

#ifndef __MSGCON_H
#define __MSGCON_H

class MessageConnection : public Connection {
    ConnectionCategory    Catg;

```

```

public:
    MessageConnection( ObjectId, ObjectId, ConnectionCategory );

    ConnectionCategory    RetCatg();
};

#endif

#ifndef __INSTCONT_H
#define __INSTCON_H

class InstanceConnection : public ConstrainedConnection {
    // no member fields
public:
    InstanceConnection( ObjectId, ObjectId,
                        int fval1 =Constraint::NOT_APPLY,
                        int fval2 =Constraint::NOT_APPLY,
                        int tval1 =Constraint::NOT_APPLY,
                        int tval2 =Constraint::NOT_APPLY );
};

#endif

```

```

/***** Class implementations for *****/
/***** Attribute and Service *****/

#ifndef __ATTRSERV_H
#define __ATTRSERV_H

#include <string.h>

class Attribute {
    char *Name,
        *Type, /* attribute type */
        *Seman; /* attribue semantics */

public:
    Attribute( char * =NULL );
    ~Attribute();

    void SetTypeAndSemantics( char * =NULL, char * =NULL);

    void GetTypeAndSemantics( char *, char * );
    char *RetAttrName( char *buf ) { return strcpy( buf, Name ); }

};

class Service {
    char *Name,
        *Type, /* service type */
        *Seman; /* service semantics */

public:
    Service( char * =NULL );
    ~Service();

    void SetTypeAndSemantics( char * =NULL, char * =NULL);

    void GetTypeAndSemantics( char *, char * );
    char *RetServName( char *buf ) { return strcpy( buf, Name ); }

};

#endif

```

```

/***** Class implementation of *****/
***** Info Windows *****/

#ifndef __INFOWIN_H
#define __INFOWIN_H

const    NO_NUM = 0,
         NUM = 1;

#include    "window.h"

class InfoDisplayWindow : public Window {
public:
    InfoDisplayWindow( char * );
    void DisplayInfo( char **, int =NUM );
};

class ClassDisplayWindow : private InfoDisplayWindow {
public:
    ClassDisplayWindow( void );
    void DisplayClasses( char ** );
};

class WholePartDisplayWindow : private InfoDisplayWindow {
public:
    WholePartDisplayWindow( void );
    void DisplayWholePartConnect( char ** );
};

class InstanceConDisplayWindow : private InfoDisplayWindow {
public:
    InstanceConDisplayWindow( void );
    void DisplayInstanceConnect( char ** );
};

class AttributeDisplayWindow : private InfoDisplayWindow {
public:
    AttributeDisplayWindow( char * );
    void DisplayAttributes( char ** );
};

class ServiceDisplayWindow : private InfoDisplayWindow {
public:
    ServiceDisplayWindow( char * );
    void DisplayServices( char ** );
};

#endif

```

```

/***** Class implementation of *****/
/***** User Interface Window *****/

#ifndef __INTFWIN_H
#define __INTFWIN_H

#include "window.h"

class Box {
public:
    char *Heading;
    char *Content;
    int Left, Top, Right, Btm;
    int TextAttr;

    Box( char *, int, int, int, int, int, int );
    ~Box();

    KEY GetInput( void );
    char *GetContentInto( char * );
};

class InterfaceWindow : public Window {
private:
    Box **InpBoxes;
    int BCount;

    void GetBoxInputs();
    KEY GetOneBoxInput( Box * ),
        GetOKFromUser();

protected:
    void DisplayBoxes( void );

public:
    InterfaceWindow( char *, int, int, int, int, int, int, int );
    ~InterfaceWindow();

    void AddBoxToInterface( Box *, int );
    int GetInputCount();
    char *GetInputInto( char *, int );
    void GetInputs();
};

```

```

inline void InterfaceWindow::AddBoxToInterface(
                                Box *box_ptr, int index )
{
    InpBoxes[ index ] = box_ptr;
}

inline int InterfaceWindow::GetInputCount()
{
    return BCount;
}

inline char *InterfaceWindow::GetInputInto( char *buf, int which )
{
    return InpBoxes[ which ]->GetContentInto( buf );
}

inline void InterfaceWindow::GetInputs()
{
    DisplayBoxes();
    GetBoxInputs();
}

#endif

```



```

/***** Class implementation of *****/
***** Dialog Windows *****/

#ifndef __INTFWINS_H
#define __INTFWINS_H

#include "intfwin.h"

class SchemaNameWindow : public InterfaceWindow {
public:
    SchemaNameWindow();
};

/*****/

class ClassAddWindow : public InterfaceWindow {
public:
    ClassAddWindow();
};

class ClassNameWindow : public InterfaceWindow {
public:
    ClassNameWindow( char * );
};

class ClassDelWindow : public ClassNameWindow {
public:
    ClassDelWindow();
};

class ClassRelocateWindow : public ClassNameWindow {
public:
    ClassRelocateWindow();
};

class ClassRepaintWindow : public ClassNameWindow {
public:
    ClassRepaintWindow();
};

/*****/

static class AttributeAddDelWindow : public InterfaceWindow {
public:
    AttributeAddDelWindow( char * );
};

class AttributeAddSemanWindow : public InterfaceWindow {
public:
    AttributeAddSemanWindow();
};

class AttributeAddWindow : public AttributeAddDelWindow {

```

```

public:
    AttributeAddWindow();
};

class AttributeDelWindow : public AttributeAddDelWindow {
public:
    AttributeDelWindow();
};

class AttrBrowseWindow : public ClassNameWindow {
public:
    AttrBrowseWindow();
};

/*****/

static class ServiceAddDelWindow : public InterfaceWindow {
public:
    ServiceAddDelWindow( char * );
};

class ServiceAddSemanWindow : public InterfaceWindow {
public:
    ServiceAddSemanWindow();
};

class ServiceAddWindow : public ServiceAddDelWindow {
public:
    ServiceAddWindow();
};

class ServiceDelWindow : public ServiceAddDelWindow {
public:
    ServiceDelWindow();
};

class ServBrowseWindow : public ClassNameWindow {
public:
    ServBrowseWindow();
};

/*****/

static class GenSpecAddDelWindow : public InterfaceWindow {
public:
    GenSpecAddDelWindow( char * );
};

class GenSpecHyAddWindow : public GenSpecAddDelWindow {
public:
    GenSpecHyAddWindow();
};

class GenSpecHyDelWindow : public GenSpecAddDelWindow {

```

```

public:
    GenSpecHyDelWindow();
};

class GenSpecLatAddWindow : public GenSpecAddDelWindow {
public:
    GenSpecLatAddWindow();
};

class GenSpecLatDelWindow : public GenSpecAddDelWindow {
public:
    GenSpecLatDelWindow();
};

/*****/

class InstConAddWindow : public InterfaceWindow {
public:
    InstConAddWindow();
};

class InstConSelectWindow : public InterfaceWindow {
public:
    InstConSelectWindow( char * );
};

class InstConDelWindow : public InstConSelectWindow {
public:
    InstConDelWindow();
};

class InstConBrowseWindow : public InstConSelectWindow {
public:
    InstConBrowseWindow();
};

/*****/

static class MesgConAddDelWindow : public InterfaceWindow {
public:
    MesgConAddDelWindow( char * );
};

class MesgConAddWindow : public MesgConAddDelWindow {
public:
    MesgConAddWindow();
};

class MesgConDelWindow : public MesgConAddDelWindow {
public:
    MesgConDelWindow();
};

```

```

/*****/

class WholePartAddWindow : public InterfaceWindow {
public:
    WholePartAddWindow();
};

class WholePartSelectWindow : public InterfaceWindow {
public:
    WholePartSelectWindow( char * );
};

class WholePartDelWindow : public WholePartSelectWindow {
public:
    WholePartDelWindow();
};

class WholePartBrowseWindow : public WholePartSelectWindow {
public:
    WholePartBrowseWindow();
};

/*****/

class DrgShtExpandWindow : public InterfaceWindow {
public:
    DrgShtExpandWindow( int, int );
};

#endif

```

```

/***** Class implementations of *****/
*****/
Menu Window, Help Window
*****/
Report Window, and Error Window *****/

#ifndef __MHEWIN_H
#define __MHEWIN_H

#include "window.h"

/***** Menu Window *****/

class MenuWindow : public Window {
    struct text_info WindSetSave; // for saving previous window settings
public:
    MenuWindow( void );
    void DisplayMenu( char * );
    friend void MenuDisplay( char * ); // prototype of this is in COMMON.H
};

/***** Help Window & Report Window *****/

class MessageWindow: public Window {
public:
    MessageWindow( char * );
};

class HelpWindow : public MessageWindow {
    struct text_info WindSetSave; // for saving previous
                                // window settings
public:
    HelpWindow( void );
    void DisplayMessage( char * );
    friend void HelpMessage( char * ); // prototype of
                                    this is in COMMON.H
};

class ReportWindow : public MessageWindow {
    struct text_info WindSetSave; // for saving previous

```

```
public:  
    ReportWindow( void );  
    void DisplayMessage( char * );  
    friend void ReportMessage( char * ); // prototype of  
                                         // this is in COMMON.H  
};  
  
/***** Error Window *****/  
  
class ErrorWindow : public Window {  
    struct text_info WindSetSave; // for saving previous  
                                   // window settings  
  
public:  
    ErrorWindow( void );  
    void DisplayMessage( char * );  
    friend void ErrorMessage( char * ); // prototype of  
                                         // this is in COMMON.H  
};  
  
#endif
```

```

/***** Class implementation of *****/
***** Window *****/

#ifndef __WINDOW_H
#define __WINDOW_H

#include "common.h"
#include "ooa.h"

#include <conio.h>

enum LineType {
    NO_LINE = 0,
    SINGLE_LINE = 1,
    DOUBLE_LINE = 2
};

class Window {
protected:
    Location Cur;

    int Left, // x1 coordinate of the screen chunk
        Top, // y1 coordinate of the screen chunk
        Right, // x2 coordinate of the screen chunk
        Btm; // y2 coordinate of the screen chunk
    int TextAttr; // text attribute information
    char *Title; // title of the window

    char *WinDispSave;
    struct text_info WinInfoSave;

    void GetTextInfo( struct text_info * ),
        SetTextInfo( struct text_info * );

public:

    Window(char *, int,int,int,int, int,int);
    ~Window();

    virtual void OutLine( LineType );

    int MoveCurLeft( unsigned ),
        MoveCurRight( unsigned ),
        MoveCurUp( unsigned ),
        MoveCurDown( unsigned );

    void SetCurTo( unsigned, unsigned ),
        MoveCurTo( Location * ),
        ResetCur( void );

```

```
void SetFgcBgc(int, int),
    SetFgcBgc(void),
    SetTitle( char * ),
    Clear( void );

};

inline void Window::SetFgcBgc()
{
    textattr( TextAttr );
}

inline void Window::Clear()
{
    clrscr();
}

#endif
```



```

/***** Class implementation of *****/
***** Message *****/

#ifndef __MESSAGE_H
#define __MESSAGE_H

#include <string.h>

class Message {
public:
    enum Result {
        Failure = 1, Success = 2
    } Status;

    char *Msg;

    Message( Result, char * =NULL );
    ~Message();
};

#endif

```

Appendix-C

This Schema Store file for the our case study problem
of Students Registration system.

```

6 6
[ 0 Person 1 (28,2)
    2
        # 0 1 4 (33,9) (33,11) (20,11) (20,14) #
        # 0 2 4 (38,9) (38,11) (56,11) (56,14) #
    0
    0
    0
    0
    3      Address Name Dept
    0
]
[ 1 Student 2 (11,15)
    2
        # 0 1 4 (33,9) (33,11) (20,11) (20,14) #
        # 1 3 4 (20,22) (20,24) (16,24) (16,28) #
    0
    1
        # 1 2 1 2 (28,16) (49,16) #
    0
    2
        # 1 2 1 2 0 4 2 (28,18) (49,18) #
        # 1 4 0 10 1 -1 4 (28,20) (31,20) (31,25) (39,25) #
    3      Roll_No Programme Fin_assistance
    1      Take_guide
]
[ 2 Instructor 2 (50,15)
    1
        # 0 2 4 (38,9) (38,11) (56,11) (56,14) #
    0
    1
        # 1 2 1 2 (28,16) (49,16) #
    0
    2
        # 1 2 1 2 0 4 2 (28,18) (49,18) #
        # 2 5 0 3 1 -1 4 (67,18) (84,18) (84,34) (80,34) #
    1      Initials
    2      Guide_student Offer_course
]
[ 3 Regd_student 2 (11,29)
    1
        # 1 3 4 (20,22) (20,24) (16,24) (16,28) #
    0
    2
        # 3 5 4 2 (28,34) (62,34) #
        # 3 4 1 4 (28,30) (32,30) (32,26) (39,26) #
    0
    0
    2      Regd_date acad_year
    3      Take_courses Pay_dues Drop_course
]
[ 4 Dues 2 (40,24)
    0
    0
    1

```

```

# 3 4 1 4 (28,30) (32,30) (32,26) (39,26) #
0
1
# 1 4 0 10 1 -1 4 (28,20) (31,20) (31,25) (39,25) #
2 Amount Due_to
1 Recv_payment
}
[ 5 Course 2 (63,33)
0
0
1
# 3 5 4 2 (28,34) (62,34) #
0
1
# 2 5 0 3 1 -1 4 (67,18) (84,18) (84,34) (80,34) #
3 Course_no Title Units
1 Validate_course
}

```

Appendix-D

This is Schema Information file for our case study problem of Students Registration system.

6

[Person

3

Address String / Address of the person /

Name String / Name of the person /

Dept String / the department to which he belongs /

0

]

[Student

3

Roll_No Long_integer / Roll no of the student /

Programme String / his programme of study /

Fin_assistance String / Financial assistance he is getting /

1

Take_guide / void Take_guide(Instructor) // takes an instructor as his guide /

]

[Instructor

1

Initials String / Instructor's name initials /

2

Guide_student / void Guide_student(Student) // guides a student /

Offer_course / void Offer_course(Course) // offer a course for this semester /

]

[Regd_student

2

Regd_date String / date on which he registered for this semester /

acad_year String / the academic year of registration date /

3

Take_courses / void Take_courses(Course *) // takes the courses for this semester /

Pay_dues / void Pay_dues(Amount) // clear the dues of the last semester /

Drop_course / void Drop_course(Instructor, Course) // drops a registered course /

]

[Dues

```

2
# Amount Amount / due amount of a student / #
# Due_to String / due to whom / #

1
# Recv_payment / void Recv_payment( Student, Amount ) // receive the payment from the student / #
]

[ Course
3
# Course_no String / code no of the course / #
# Title String / title of the course / #
# Units String / the units for the course / #

1
# Validate_course / void Validate_course( Course ) // check the validity of the course / #
]

```

Appendix-E

This is Schema Templates file for our case study problem
of Students Registration System.

/****** The class templates for the schema are as follows *****/

```
class Person {
private:
    String      Address;
    String      Name;
    String      Dept;

};

class Student : public Person {
private:
    Instructor   *instance_to_Instructor;
    Dues         *instance_to_Dues;

    Instructor   **message_to_Instructor;

    Long_integer Roll_No;
    String        Programme;
    String        Fin_assistance;

    void Take_guide( Instructor ) ;

};

class Instructor : public Person {
private:
    Student       *instance_to_Student;
    Course        *instance_to_Course;

    Student       **message_to_Student;

    String        Initials;

    void Guide_student( Student ) ;
    void Offer_course( Course ) ;

};

class Regd_student : public Student {
private:
    Course **message_to_Course;
    Dues   **message_to_Dues;

    String      Regd_date;
    String      acad_year;

    void Take_courses( Course * ) ;
    void Pay_dues( Amount ) ;
    void Drop_course( Instructor, Course ) ;

};
```

```

class Dues {
private:
    Student      *instance_to_Student;

    Regd_student *message_to_Regd_student;

    Amount      Amount;
    String      Due_to;

    void Recv_payment( Student, Amount ) ;
};

```

```

class Course {
private:
    Instructor      *instance_to_Instructor;

    Regd_student    *message_to_Regd_student;

    String          Course_no;
    String          Title;
    String          Units;

    void Validate_course( Course ) ;
};

```

```

/***** This is the end of the file *****/

```

Appendix-F

This is Analysis Document file for our case study problem
of Students Registration System.

/***** The Document for the schema follows *****/

CLASS NAME :: Person

USAGE :: Only as class

ATTRIBUTES ::

NAME :: Address

TYPE :: String

SEMANTICS ::

Address of the person

NAME :: Name

TYPE :: String

SEMANTICS ::

Name of the person

NAME :: Dept

TYPE :: String

SEMANTICS ::

the department to which he belongs

PUBLIC INTERFACE ::

Services are absent

DERIVED FROM CLASSES ::

No other class

COMPOSED OF CLASSES ::

No other classes

ASSOCIATED WITH ::

No other class

COMMUNICATES WITH ::

No other class

END OF THE CLASS Person

CLASS NAME :: Student

USAGE :: Both as class and object

ATTRIBUTES ::

NAME :: Roll_No

TYPE :: Long_integer

SEMANTICS ::

Roll no of the student

NAME :: Programme

TYPE :: String

SEMANTICS ::

his programme of study

NAME :: Fin_assistance

TYPE :: String

SEMANTICS ::

Financial assistance he is getting

```

PUBLIC INTERFACE ::
    NAME :: Take_guide
    PROTOTYPE ::
        void Take_guide( Instructor )
    SEMANTICS ::
        takes an instructor as his guide

DERIVED FROM CLASSES ::
    Person

COMPOSED OF CLASSES ::
    No other classes

ASSOCIATED WITH ::
    Instructor
    Dues

COMMUNICATES WITH ::
    Instructor

END OF THE CLASS Student

CLASS NAME :: Instructor

    USAGE :: Both as class and object

    ATTRIBUTES ::
        NAME :: Initials
        TYPE :: String
        SEMANTICS ::
            Instructor's name initials

    PUBLIC INTERFACE ::
        NAME :: Guide_student
        PROTOTYPE ::
            void Guide_student( Student )
        SEMANTICS ::
            guides a student
        NAME :: Offer_course
        PROTOTYPE ::
            void Offer_course( Course )
        SEMANTICS ::
            offer a course for this semester

    DERIVED FROM CLASSES ::
        Person

    COMPOSED OF CLASSES ::
        No other classes

    ASSOCIATED WITH ::
        Student
        Course

    COMMUNICATES WITH ::
        Student

END OF THE CLASS Instructor

CLASS NAME :: Regd_student

    USAGE :: Both as class and object

    ATTRIBUTES ::

```

```

NAME :: Regd_date
TYPE :: String
SEMANTICS ::
    date on which he registered for this semester
NAME :: acad_year
TYPE :: String
SEMANTICS ::
    the academic year of registration date

PUBLIC INTERFACE ::
NAME :: Take_courses
PROTOTYPE ::
    void Take_courses( Course * )
SEMANTICS ::
    takes the courses for this semester
NAME :: Pay_dues
PROTOTYPE ::
    void Pay_dues( Amount )
SEMANTICS ::
    clear the dues of the last semester
NAME :: Drop_course
PROTOTYPE ::
    void Drop_course( Instructor, Course )
SEMANTICS ::
    drops a registered course

DERIVED FROM CLASSES ::
    Student

COMPOSED OF CLASSES ::
    No other classes

ASSOCIATED WITH ::
    No other class

COMMUNICATES WITH ::
    Course
    Dues

END OF THE CLASS Regd_student

CLASS NAME :: Dues

USAGE :: Both as class and object

ATTRIBUTES ::
NAME :: Amount
TYPE :: Amount
SEMANTICS ::
    due amount of a student
NAME :: Due_to
TYPE :: String
SEMANTICS ::
    due to whom

PUBLIC INTERFACE ::
NAME :: Recv_payment
PROTOTYPE ::
    void Recv_payment( Student, Amount )
SEMANTICS ::
    receive the payment from the student

DERIVED FROM CLASSES ::
    No other class

COMPOSED OF CLASSES ::

```

ASSOCIATED WITH ::
Student

COMMUNICATES WITH ::
Regd_student

END OF THE CLASS Dues

CLASS NAME :: Course

USAGE :: Both as class and object

ATTRIBUTES ::
NAME :: Course_no
TYPE :: String
SEMANTICS ::
code no of the course
NAME :: Title
TYPE :: String
SEMANTICS ::
title of the course
NAME :: Units
TYPE :: String
SEMANTICS ::
the units for the course

PUBLIC INTERFACE ::
NAME :: Validate_course
PROTOTYPE ::
void Validate_course(Course)
SEMANTICS ::
check the validity of the course

DERIVED FROM CLASSES ::
No other class

COMPOSED OF CLASSES ::
No other classes

ASSOCIATED WITH ::
Instructor

COMMUNICATES WITH ::
Regd_student

END OF THE CLASS Course

/***** This is the end of the Document *****/